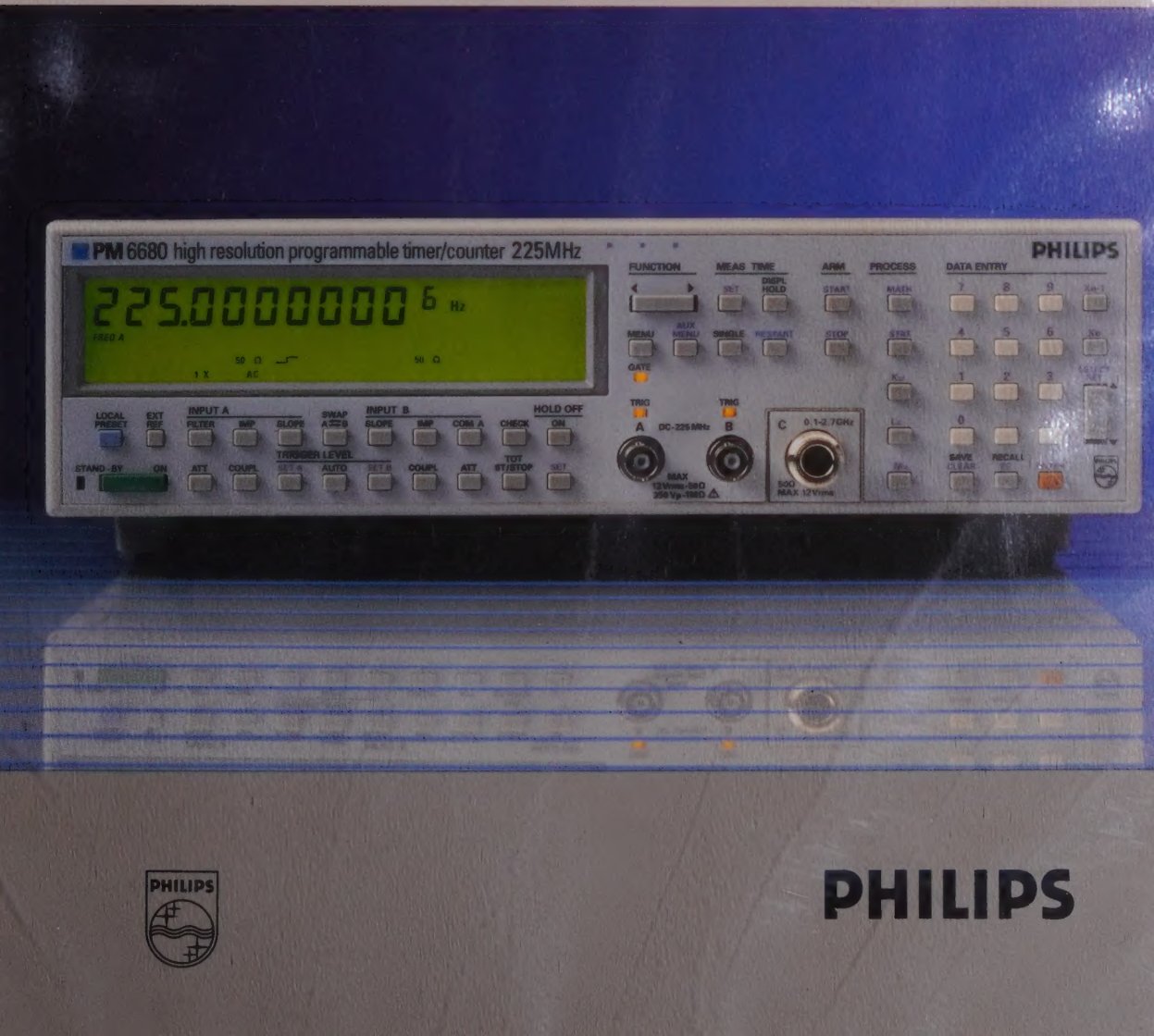
PM 6680
500 ps/225 MHz TIMER/COUNTER
Programming Manual



PHILIPS

PHILIPS

Table of contents

1 Getting Started

Finding Your Way Through This Manual .. 1-2
Sections 1-2
Yellow Pages 1-3
Index 1-3
Manual Conventions 1-4
Syntax Specification Form 1-4
Example language 1-5
Setting Up the Instrument 1-6
Setting the GPIB address 1-6
Power-on 1-6
Testing the bus 1-6
Interface Functions 1-7
What can I do with the bus? 1-7

2 Bus Commands for the Benchtop User

Commands for front panel keys 2-2
Commands for measuring functions 2-4
Commands for rear panel functions 2-5
Default settings (after *RST) 2-6

3 Introduction to SCPI

What is SCPI? 3-2
Reason for SCPI 3-2
Compatibility 3-3
Management and maintenance of programs 3-3
How does SCPI Work in the Instrument? . 3-4
Message Exchange Control protocol 3-5
Protocol requirements 3-6
Device clear 3-7
Program and Response Messages 3-10
Syntax and Style 3-11
Syntax of Program Messages 3-11
Syntax of Response Messages 3-14
Command Tree 3-16
Moving down the Command Tree 3-17
Parameters 3-18
Numeric Data 3-18
Booleans 3-19
Expression Data 3-19
Other data types 3-20
Summary 3-20
Status Reporting System 3-21
Structure of the Status System 3-21
Error Reporting 3-23
Initialization and resetting 3-24
Reset strategy 3-24

4 Instrument Model

Introduction 4-2
Measurement Function Block 4-3
Other subsystems 4-4
About all commands 4-5
MEASurement function 4-5

5 Analog Output Subsystem

Analog Output Subsystem Command Tree 5-2
Enable Analog Out 5-3
Scaling Factor, Analog Output 5-4
How to use the Analog Output 5-5

6 Calculate Subsystem

Calculate Subsystem Command Tree 6-2
Enable Calculation 6-3
Read Calculated Data 6-4
Recalculate Data 6-5
Select Mathematical Expression 6-6
Enable Mathematics 6-7
Enable Statistics 6-8
Statistical Type 6-9
Sample Size for Statistics 6-10
Enable Monitoring of Parameter Limits . 6-11

Set Upper Limit 6-12
Check Against Upper Limit 6-13
Set Low Limit 6-14
Check Against Lower Limit 6-15
Limit Fail 6-16
Query syntax: 6-16

7 Calibration Subsystem

Calibration Subsystem Command Tree .. 7-2
Calibration of Interpolator 7-3

8 Display Subsystem

Display Subsystem Command Tree 8-2
Display State 8-3

9 Format Subsystem

Format Subsystem Command Tree 9-2
Response Data Type 9-3
Response Data Format 9-4

10 Input Subsystems

Input Subsystems Command Tree 10-2
Input A 10-2
Input B 10-2
Input E 10-2
Attenuation 10-3
AC/DC Coupling 10-4
Input Impedance 10-5
Fixed Trigger Level 10-6
Autotrigger 10-7
Trigger Slope 10-8
Low Pass Filter 10-9
Common via A 10-10
Using the Input settings 10-11

11 Measurement Function

Measure Function Command Trees 11-2
Measure once 11-4
Measure Array 11-5
Configure 11-6
Configure Array 11-7
Read one Result 11-8
Read Array 11-9
Fetch one Result 11-10
Fetch an Array of Results 11-11

Explanations of the Measuring Functions 11-13
Frequency 11-14
Frequency Ratio 11-15
Burst Carrier Frequency 11-16
Pulse Repetition Frequency 11-17
Period 11-18
Time Interval 11-19
Phase 11-20
Pulse Width 11-21
Duty Cycle 11-22
Transition Time 11-23
Voltage 11-24
Totalize X gated by Y 11-25
Totalize X gated by Y, accumulated ... 11-26
Totalize X start/stop by Y 11-27
Totalize X-Y During a Preset Time 11-28
Totalize X–Y Manually 11-29
Setting up totalizing 11-30
Intermediate results 11-30
How to use the Measure function 11-31
MEASure? 11-31
CONFigure; READ? 11-31

12 Memory Subsystem

Memory Subsystem Command Tree 12-2
Delete one Macro 12-3
Memory Free for Macros 12-4
Memory states 12-5
Save 12-6
Recall 12-7
Learn Device Setup 12-8
Define Macro 12-9
Enable Macros 12-10
Get Macro 12-11
Learn Macro 12-12
Purge Macros 12-13
How to use Macros 12-15
Define macro command 12-16
Deleting macros 12-16
Enabling and disabling macros 12-17
How to execute a macro? 12-17
Retrieve a macro 12-18

13 Sense Command Subsystems

Sense Subsystem Command Tree 13-2
Input channels 13-3
Select Measuring Function 13-4
Set the Measuring-Time 13-5
Hold Off on/off 13-6
Hold Off Mode 13-7
Hold Off Time 13-8
Hold Off Events 13-9
Resolution 13-10
Average or Single? 13-11
Average Samples 13-12
Average Mode 13-13
High Speed Voltage Measurements ... 13-14
Select Oscillator Source 13-15
Gate on/off 13-16
Gated Voltage Measurement 13-17
Internal Format 13-18
How to Select Input 13-19

14 Status Subsystem

Status Subsystem Command Tree 14-2
Operation Status, Condition 14-3
Operation Status, Enable 14-4
Operation Status, Event 14-5
Questionable Data/signal, Condition 14-6
Questionable Data/signal, Enable 14-7
Questionable Data/signal, Event 14-8
Device Status Register, Enable 14-9
Device Status Register, Event 14-10
Enable Device Status Reporting 14-11
Clear Status Command 14-12
Standard Event Status Enable 14-13
Event Status Register 14-14
Power-on Status Clear 14-15
Service Request Enable 14-16
Status Byte Query 14-17
How to use Status Reporting 14-19
Standard event status register 14-24
Device defined status registers 14-26
Status of the output queue 14-33
Status of the error message queue .. 14-33

15 Synchronization

Operation Complete 15-2
Operation Complete Query 15-3
Wait-to-continue 15-4

16 System Subsystem

System Subsystem Command Tree 16-2
GPIB Address 16-3
Error Description 16-4
Key 16-5
Preset 16-6
Read or Send Settings 16-7
Timeout ON/OFF 16-8
Timeout, Set 16-9
Unprotect 16-10
System Version 16-11
Identification query 16-12
Option Identification 16-13
Protected User Data 16-14
Reset 16-15

17 Test Subsystem

Test Subsystem Command Tree 17-2
Select Check signal 17-3
Select Self-tests 17-4
Self Test 17-5

18 Trigger/Arming Subsystem

Trigger Subsystem Command Tree 18-2
Initiate Measurement 18-3
Continuously Initiated 18-4
Abort Measurement 18-5
Bus Arming Exit 18-6
Bus Arming on/off 18-7
No. of Measurements on each Bus arm 18-8
Delay after External Start Arming 18-9
External Events before Start Arming ... 18-10
External Arming Start Slope 18-11
External Arming Start Source 18-12
No. of Triggerings on each Ext Arm start 18-13
Delay after External Stop Arming 18-14
External Events before Stop Arming ... 18-15
External Stop Arming Slope 18-16
External Stop Arming Source 18-17
Trigger 18-18
How to use the Trigger System 18-19
Instrument action 18-19

Triggering 18-22

19 How to Measure

Introduction 19-2
Controller synchronization 19-2
Some basic PM 6680 Commands 19-3
CONFigure 19-5
INITiate 19-5
INITiate :CONTinuous 19-6
ABORt 19-6
FETCh? 19-6
READ? 19-6
MEASure? 19-7
Basic Measurement Method 19-8
Individually Synchronized Measurements 19-8
Block Synchronized Measurements 19-9
General Speed Improvements 19-10
Speed summary 19-13

20 Programming Examples

Introduction 20-2
GW-Basic examples for PM 2201 ... 20-3
Setting up the interface 20-4
Example 1, Statistics 20-6
Example 2, Limit testing 20-8
Example 4, Frequency profiling 20-10
Example 5, Fast sampling 20-12
Example 6, Status Reporting 20-14
'C' examples for PM 2201 20-17
Example 1, Statistics 20-18
Example 2, Limit testing 20-20
Example 3, REAL data format 20-22
Example 4, Frequency profiling 20-24
Example 5, Fast sampling 20-26
GW-Basic examples for National Instruments PC-IIA 20-29
Setting up the interface 20-30
Example 1, Statistics 20-31
Example 2, Limit testing 20-33
Example 4, Frequency profiling 20-34
Example 5, Fast sampling 20-36
Example 6, Status Reporting 20-38
'C' examples for National Instruments PC-IIA 20-41
Example 1, Statistics 20-42
Example 2, Limit testing 20-44
Example 3, REAL data format 20-46
Example 4, Frequency profiling 20-48
Example 5, Fast sampling 20-50

21 Error Messages

How to use Error Reporting 21-19

22 Appendix

Appendix 1, Alphanumeric Command List 22-3

23 Index

Chapter 1

Getting Started

Finding Your Way Through This Manual

You should use this Programming Manual together with the PM 6680 Operators Manual. That manual contains specifications for the timer/counter and explanations of the possibilities and limitations of the different measuring functions. It also contains the installation procedure for the interface, if you bought it separately.

Sections

The chapters in this manual are divided into three sections aimed at different levels of reader knowledge.

The 'General' section, that can be disregarded by the users who knows the IEEE-488 and SCPI standards:

Chapter 2 *Bus Commands for the Benchtop User* gives bus commands for the front panel keys.

Chapter 3 *Introduction to SCPI* explains syntax data formats, status reporting, etc.

The 'Programmers Reference' section of this manual contains:

Chapter 4, *Instrument Model* explains how the instrument looks from the bus. This instrument is not quite the same as the one used from the front panel.

Chapter 5 to 18, *Subsystems,* This is the "Command reference" that gives complete information on all commands. The subsystems are sorted alphabetically.

Chapter 21, *Error Messages* contains a list of all error messages that can be generated during bus control.

The Practical Section of this manual contains:

Chapter 19, *How to Measure* is a set of measuring situations which the user is often confronted with when programming a timer/ counter.

Chapter 20, *Programming Examples*, contains examples of typical programs for a wide variety of applications. These programs are written in GW-basic and C. They can be used as they are on an IBM PC-compatible controller with a PM 2201 interface board.

Yellow Pages

The yellow pages at the end of the more complex subsystem descriptions, contain information about how to use the subsystem.

Index

You can also use the alphabetical command list in the appendix, in addition to the index, to get an overview of the commands. The index is also useful when looking for additional information on the command you are currently working with.

Manual Conventions

Syntax Specification Form

This manual uses the EBNF (Extended Backus-Naur Form) notation for describing syntax. This notation uses the following types of symbols:

- *Printable characters:* Printable characters such as Command headers, etc., are printed just as they are, e.g., Period means that you should type the word PERIOD.

  The following printable characters have a special meaning and will only be used in that meaning: # ' " ( ) : ; *
  Read' Introduction to SCPI' chapter for more information.

- *Non-printable characters:* Two non-printable characters are used: ␣ indicates the space character (ASCII code 32) and ↵ indicates the new line character (ASCII code 10).

- *Specified expressions: < >* Symbols and expressions that are further specified elsewhere in this manual are placed between the <> signs.
  For example <Dec. data.>. The following explanation is found on the same page: "Where <Dec. data> is a four-digit number between 0.1 and $8 \cdot 10^{-9}$.

- *Alternative expressions giving different result: |* Alternative expressions giving different results are separated by |. For example On|Off means that the function may be switched on or off.

- *Grouping: « »* Example: FORMat␣«ASCII|REAL» specifies the command header FORMat followed by a space character and either ASCII or REAL.

- *Optionality: [ ]* An expression placed within [ ] is optional.

  Example: [:VOLT] :FREQuency

  means that command FREQuency may or may not be preceded by :VOLT.

- *Repetition: { }* An expression placed within {} can be repeated zero or more times.

- *Equality: =* Equality is specified with = Example: <Separator> = ,

Mnemonic conventions

● *Truncation rules*

All commands can be truncated to short forms. The truncation rules are as follows:

- The short form is the first four characters of the command.
- If the fourth character in the command is a vowel, then the short form is the first three characters of the command. This rule is not used if the command is only four characters.
- If the last character in the command is a digit, then this digit is appended to the short form.

Examples:

Longform	Shortform
:MEASURE	:MEAS
:NEGATIVE	:NEG
:DREGISTER0	:DREG0
:EXTERNAL2	:EXT2

The short form is always printed in CAPITALS in this manual: :MEASure, :NEGative, :DREGister0, :EXTernal2 etc.

Example language

Small examples are given at various places in the text. These examples are not in BASIC or C, nor are they written for any specific controller. They only contain the characters you should send to the timer/counter and the responses that you should read with the controller.

Example:

SEND→ `MEAS:FREQ?`

This means that you should program the controller so that it addresses the counter and outputs this string on the GPIB.

READ← `1.234567890E6`

This means that you should program the controller so that it can receive this data from the GPIB, then address the counter and read the data.

Setting Up the Instrument

Setting the GPIB address

The address switches on the rear panel of the PM 6680 are set to 10 when the counter is delivered. If you want to use another bus address, you can set these switches to any address between 0 and 30 as shown in the following table.

Address	Switch Settings	Address	Switch Settings	Address	Switch Settings
0	00000	**10**	**01010**	20	10100
1	00001	11	01011	21	10101
2	00010	12	01100	22	10110
3	00011	13	01101	23	10111
4	00100	14	01110	24	11000
5	00101	15	01111	25	11001
6	00110	16	10000	26	11010
7	00111	17	10001	27	11011
8	01000	18	10010	28	11100
9	01001	19	10011	29	11101
				30	11110

Power-on

When you switch on PM 6680 it restores the setting it had when switched off because of the battery back-up of the internal memory.

● *Stand-by*

When the counter is in REMOTE mode you cannot switch it off. You must first enable Local control by pressing LOCAL.

Testing the bus

To test that the instrument is operational via the bus, use ✱IDN? to identify the instrument and ✱OPT? to identify which options are installed. (See chapter 16, 'System Subsystem' , ✱IDN? and ✱OPT?)

Interface Functions

What can I do with the bus?

All the capabilities of the interface for the PM 6680 are explained below. If you want a complete description of all GPIB interface functions, consult the Philips Instrumentation Systems Reference Manual available from your Philips representative.

- *Summary*

Description	Code
Source handshake	SH1
Acceptor handshake	AH1
Control function	C0
Talker Function	T6
Listener function	L4
Service request	SR1
Remote/local function	RL1
Parallel poll	PP0
Device clear function	DC1
Device trigger function	DT1
Bus drivers	E2

- *SH1 and AH1* simply means that the timer/counter can exchange data with other instruments or a controller, using the bus handshake lines; DAV, NRFD, NADC.

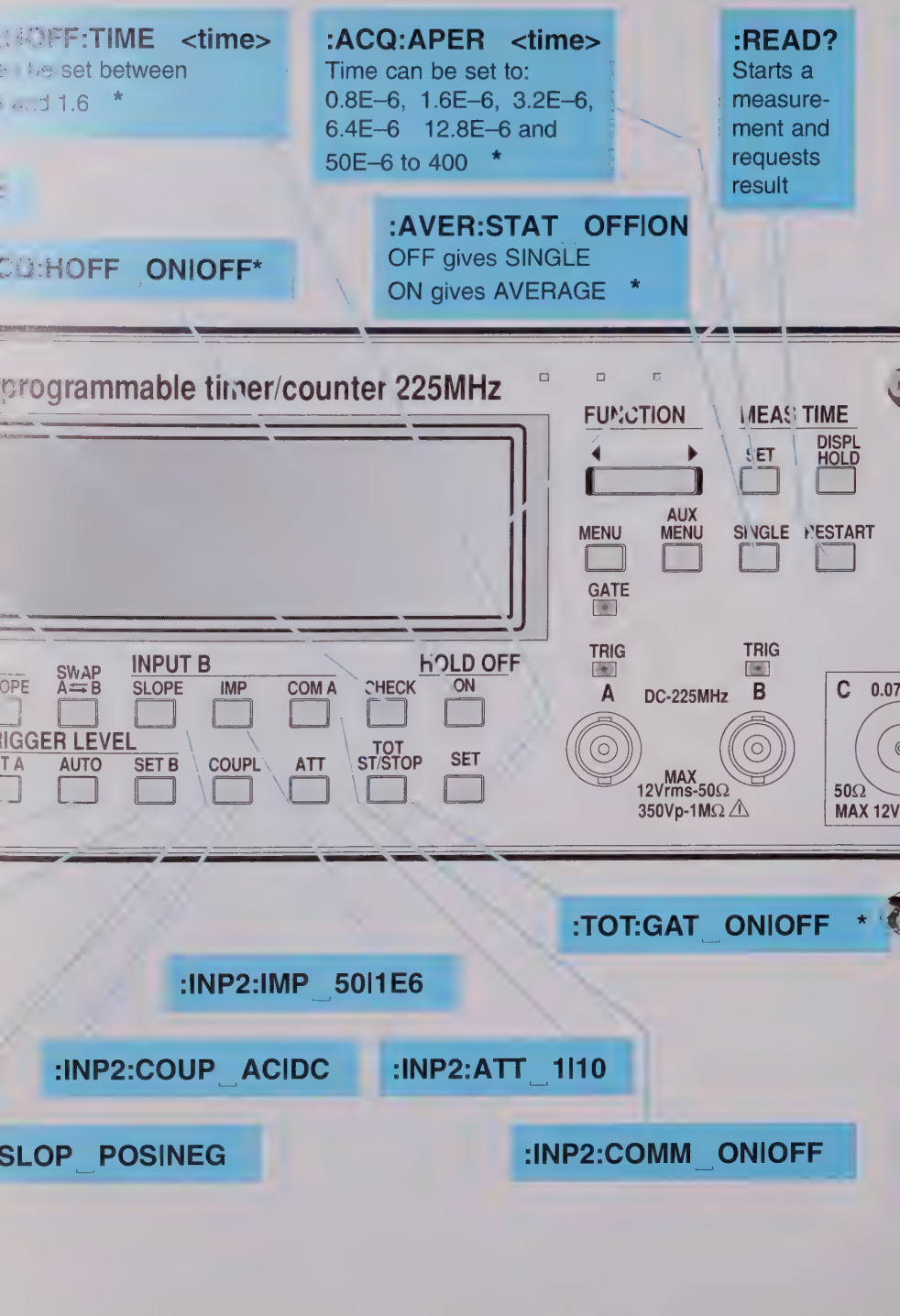
- *Control function, C0* The timer/counter does not function as a controller.

- *Talker function, T6* The timer/counter can send responses and the results of its measurements to other devices or to the controller. T6 means that it has the following functions:
  - Basic talker.
  - No talker only.
  - It can send out a status byte as response to a serial poll from the controller.
  - Automatic un-addressing as a talker when it is addressed as a listener.

- *Listener function, L4* — The timer/counter can receive programming instructions from the controller. L4 means the following functions:
  - Basic listener.
  - No listen only.
  - Automatic un-addressing as listener when addressed as a talker.
- *Service request, SR1* — The timer/counter can call for attention from the controller, e.g. when a measurement is completed and a result is available.
- *Remote/local, RL1* — You can control the timer/counter manually (locally) from the front panel, or remotely from the controller. The LLO, local-lock-out function, can disable the LOCAL button on the front panel.
- *Parallel poll, PP0* — The timer/counter does not have any parallel poll facility.
- *Device clear, DC1* — The controller can reset the timer/counter via interface message DCL (Device clear) or SDC (Selective Device Clear).
- *Device Trigger, DT1* — You can start a new measurement from the controller via interface message GET (Group Execute Trigger).
- *Bus drivers, E2* — The GPIB interface has tri-state bus drivers.

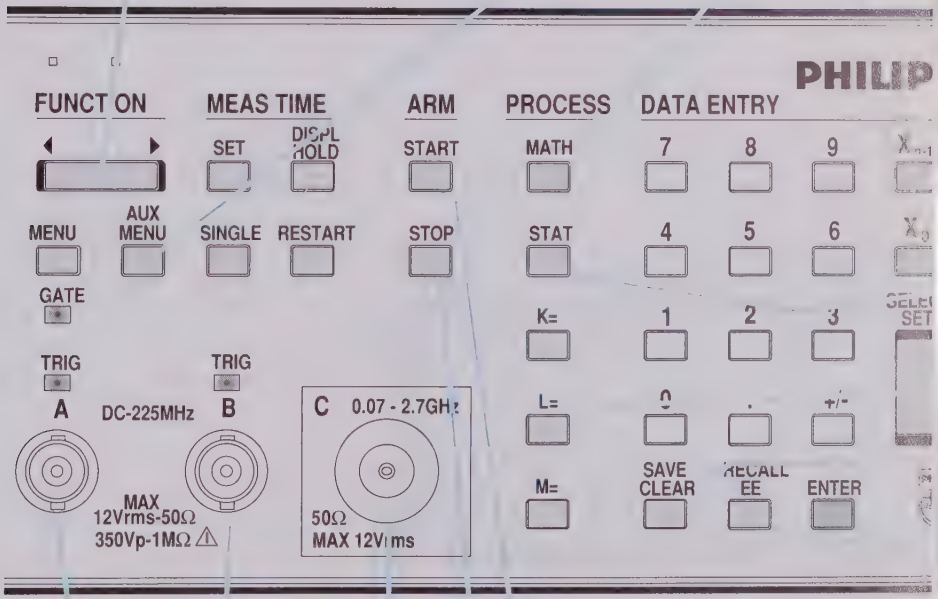
Chapter 2

Bus Commands for the Benchtop User


:OFF:TIME <time>
be set between
and 1.6 *
:ACQ:APER <time>
Time can be set to:
0.8E–6, 1.6E–6, 3.2E–6,
6.4E–6 12.8E–6 and
50E–6 to 400 *
:READ?
Starts a measure-
ment and
requests
result
:AVER:STAT OFFION
OFF gives SINGLE
ON gives AVERAGE *
CQ:HOFF ONIOFF*
programmable timer/counter 225MHz
FUNCTION
MEAS TIME
SET
DISPL
HOLD
MENU
AUX
MENU
SINGLE
RESTART
GATE
TRIG
A
TRIG
B
DC-225MHz
C 0.07
MAX
12Vrms-50Ω
350Vp-1MΩ
50Ω
MAX 12V
OPE
SWAP
A⇄B
INPUT B
SLOPE
IMP
COM A
CHECK
HOLD OFF
ON
IGGER LEVEL
T A
AUTO
SET B
COUPL
ATT
TOT
ST/STOP
SET
:TOT:GAT ONIOFF *
:INP2:IMP 50I1E6
:INP2:COUP ACIDC
:INP2:ATT 1I10
SLOP POSINEG
:INP2:COMM ONIOFF

:FUNC "function channel,channel" *
Function and channel is explained on the next page

The functions in the tree are found in ma stem command tree No. of samples for s Calculate subsystem



Input 1 Input 2 Input 3

:ARM:STOP:SOUR E
Switches on stop arming on ir
:ARM:STOP:SOUR IM
Switches off stop arming
:ARM:STOP:SLOP PO
Selects stop arming slope

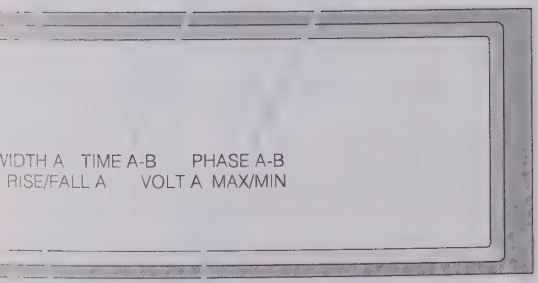
* These commands are from the SENSE subsystem

** These commands are from the TRIGGER subsystem

:FUNC "TINT 1,2"

"PWID 1"

:FUNC "PHAS 1,2"



:FUNC "VOLT:MAX 1"

:FUNC "VOLT:MIN 1"

1"

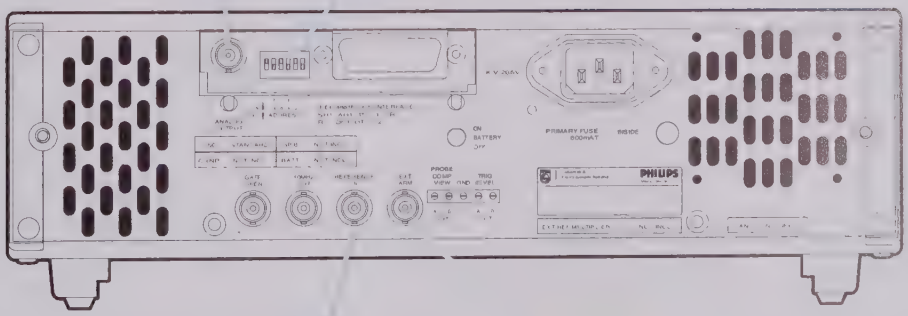
:FUNC "VOLT:PTP 1"

:FUNC "RISE|FALL:TIME 1"

All commands on this page are from the SENSE subsystem

:OUTP ON|OFF
:OUTP:SCAL <scaling factor>

:SYST:COMM:GPIB:ADDR <address>
<address can be between 1 and 30.



Input 4

:ROSC:SOUR INT|EXT *

* This command is from the SENSE subsystem

Default settings (after ✷RST)

PARAMETER	VALUE/ SETTING
Input A:	
Trigger level	AUTO
Impedance	1 MΩ
Manual Trigger level (Controlled by AUTO-trig)	0V
Manual Attenuator (Controlled by AUTO-trig)	1X
Coupling	AC
Trigger slope	Pos
Filter	OFF
Input B:	
Trigger level	AUTO
Impedance	1 MΩ
Manual Trigger level (Controlled by AUTO-trig)	0V
Manual Attenuator (Controlled by AUTO-trig)	1X
Coupling	DC
Trigger slope	Pos
Common	OFF
Arming:	
Start	OFF
Stop	OFF
Delay	Start, Time, OFF
Channel	Ext Arm Input E

PARAMETER	VALUE/ SETTING
Statistics:	
Statistics	OFF
Mathematics	OFF
Sample size in Statistics	100
Sample size in Time Interval Average	100
Mathematical constants:	
K= and M=	1
L=	0
Miscellaneous:	
Function	FREQ A
Time out	100 ms, OFF
Measuring time	0.2 s
Check	OFF
Single cycle	OFF
Analog output control	OFF
Hold Off	Time, OFF
Memory Protection (Memory 10 to19)	OFF
Auxiliary functions	All switched OFF
Blank LSD	OFF

Chapter 3

Introduction to SCPI

What is SCPI?

SCPI (Standard Commands for Programmable Instruments) is a standardized set of commands used to remotely control programmable test and measurement instruments. The PM 6680 firmware contains the SCPI. It defines the syntax and semantics that the controller must use to communicate with the instrument.

This chapter is an overview of SCPI and shows how SCPI is used in Philips Timer/Counters.

SCPI is based on IEEE-488.2 to which it owes much of its structure and syntax. SCPI can however be used with any of the standard interfaces, such as GPIB (=IEC625/ IEEE-488), VXI and RS-232.

Reason for SCPI

For each instrument function, SCPI defines a specific command set. The advantage of SCPI is that programming an instrument is only function dependent and no longer instrument dependent. Several different types of instruments, for example an oscilloscope, a counter and a multimeter, can carry out the same function, such as frequency measuring. If these instruments are SCPI compatible, you can use the same commands to measure the frequency on all three instruments, although there may be differences in accuracy, resolution, speed, etc.

Compatibility

SCPI provides two types of compatibility:

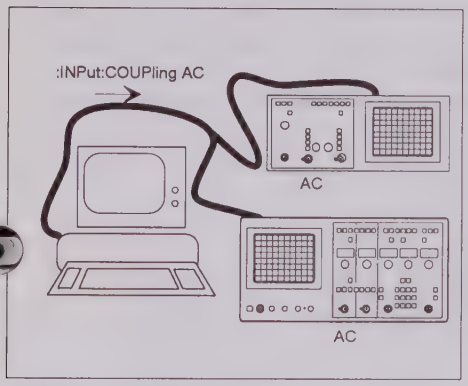


Fig. 3-1 ***Vertical***
This means that all instruments of the same type have identical controls. For example, oscilloscopes will have the same controls for timebase, triggers, and voltage settings.

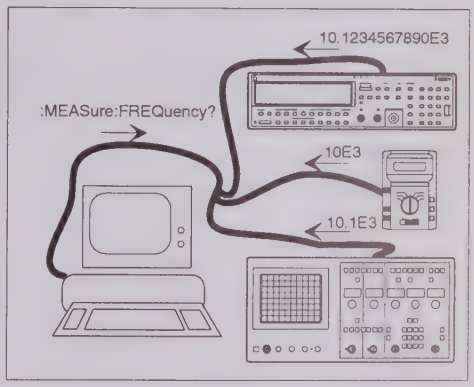


Fig. 3-2 ***Horizontal***
This means that instruments of different types that performs the same functions have the same commands. For example, a DMM, an oscilloscope, and a counter can all measure frequency with the same commands.

Management and maintenance of programs

SCPI simplifies maintenance and management of the programs. Today changes and additions in a good working program are hardly possible because of the great diversity in program messages and instruments. Programs are difficult to understand for people other than the original programmer. After some time, however, even the programmer may be unable to understand them. A programmer with SCPI experience, however, will rather quickly understand the meaning and reasons of a SCPI program, because of his knowledge of the standard. Changes, extensions and additions are much easier to make in an existing application program. SCPI is a step towards portability of instrument programming software and, as a consequence, it allows the exchange of instruments.

How does SCPI Work in the Instrument?

The functions inside an instrument that control the operation provide SCPI compatibility. Fig. 3-3 shows a simplified logical model of the message flow inside a SCPI instrument.

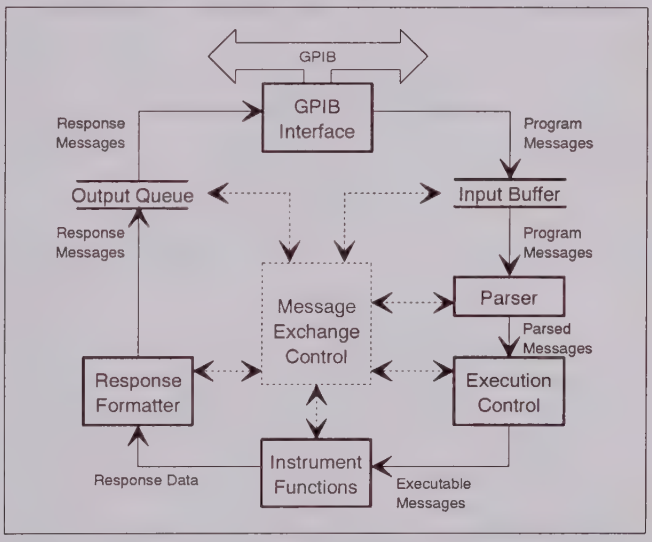


Fig.3-3 *Overview of the firmware in an SCPI instrument.*

When the controller sends a message to a SCPI instrument roughly the following happens:

- The GPIB controller addresses the instrument as listener.
- The GPIB interface function places the message in the Input Buffer.
- The Parser fetches the message from the Input Buffer, parses (decodes) the message, and checks for the correct syntax. The instrument reports incorrect syntax by sending command errors via the status system to the controller.

Moreover, the parser will detect if the controller requires a response. This is the case when the input message is a query (command with a "?" appended).

The Parser will transfer the executable messages to the Execution Control block in token form (internal codes). The Execution Control block will gather the information required for a device action and will initiate the requested task at the appropriate time. The instrument reports execution errors via the status system over the GPIB and places them in the Output Queue.

- When the controller addresses the instrument as talker, the instrument takes data from the Output Queue and sends it over the GPIB to the controller.

Message Exchange Control protocol

Another important function is the Message Exchange Control, defined by IEEE 488.2. The Message Exchange Control protocol specifies the interactions between the several functional elements that exist between the GPIB functions and the device-specific functions, see Fig. 3-3 . The Message Exchange Control protocol specifies how the instrument and controller should exchange messages. For example, it specifies exactly how an instrument shall handle program and response messages that it receives from and returns to a controller. This protocol introduces the idea of commands and queries; queries are program messages that require the device to send a response. When the controller does not read this response, the device will generate a Query Error. On the other hand, commands will not cause the device to generate a response. When the controller tries to read a response anyway, the device then generates a Query Error.

The Message Exchange Control protocol also deals with the order of execution of program messages. It defines how to respond if Command Errors, Query Errors, Execution Errors, and Device-Specific errors occur. The protocol demands that the instrument report any violation of the IEEE-488.2 rules to the controller, even when it is the controller that violates these rules.

The IEEE 488.2 standard defines a set of operational states and actions to implement the message exchange protocol. These are shown in the following table:

State	Purpose
IDLE	wait for messages
READ	Read and execute messages
QUERY	Store responses to be sent
SEND	Send responses
RESPONSE	Complete sending responses
DONE	Finished sending responses
DEADLOCK	the device cannot buffer more data

Action	Reason
Unterminated	The controller attempts to read the device without first having sent a complete query message
Interrupted	The device is interrupted by a new program message before it finishes sending a response message

Protocol requirements

In addition to the above functional elements, which process the data, the message exchange protocol has the following characteristics:

- The controller must end a program message containing a query with a message terminator before reading the response from the device (address the device as talker). If the controller breaks this rule, the device will report a query error (unterminated action).
- The controller must read the response to a query in a previously (terminated) program message before sending a new program message. When the controller violates this rule, the device will report a query error (interrupted action).
- The instrument sends only one response message for each query message. If the query message resulted in more than one answer, all answers will be sent in one response message.

● *Order of execution*

Deferred commands

Execution control collects commands until the end of the message, or until it finds a query or other special command that forces execution. It then checks that the setting resulting from the commands is a valid one: No range limits are exceeded, no coupled parameters are in conflict, etc. If this is the case, the commands are executed in the sequence they have been received; otherwise, an execution error is generated, and the commands are discarded.
This deferred execution guarantees that:

- All valid commands received before a query are executed before the query is executed.
- All queries are executed in the order they are received.
- The order of execution of commands is never reversed.

● *Sequential commands*

All commands in the PM 6680 are sequential.

Device clear

The Device Clear (DCL) or Selective Device Clear (SDC) command clears the input buffer and the output queue, re-sets the parser and clears any pending commands.

The device clear commands will not:

- Change the instrument settings or stored data in the instrument.
- Interrupt or affect any device operation in progress.
- Change the status byte register other than clearing the MAV bit as a result of clearing the output queue.

NOTE: Many older IEEE-instruments, that are not IEEE-488.2 compatible returned to the power-on default settings when receiving a device clear command. IEEE-488.2 does not allow this.

● *When to use a device clear command*

The command is also useful to escape from erroneous conditions without having to alter the current settings of the instrument. The instrument will then discard pending commands and will clear responses from the output queue. For example; suppose you are using the PM 6680 in an automated test equipment system where the controller program returns to its main loop on any error condition in the system or the tested unit. To ensure that no unread query response remains in the output queue and that no unparsed message is in the input buffer, it is wise to use device clear. (Such remaining responses and commands could influence later commands and queries.)

Remote local protocol

● *Definitions*

Remote operation

When an instrument operates in remote, all local controls, except the local key, are disabled.

Local operation

An instrument operates in local when it is not in remote mode as given above.

Local lockout

In addition to the remote state, an instrument can be set to remote with 'local lockout'. This disables the return-to-local button. In theory, the state local with local lockout is also possible; then, all local controls except the return-to-local key are active.

The PM 6680 in remote operation

When the PM 6680 is in remote operation, it disables all its local controls except the LOCAL key.

The PM 6680 in local operation

When the PM 6680 is in local operation the instrument is fully programmable both from the front panel and from the bus. If a bus message arrives while a change is being entered from the front panel, the front panel entry is interrupted and the bus message is executed.

Program and Response Messages

The communication between the system controller and the SCPI instruments connected to the GPIB takes place through Program and Response Messages. A Program Message is a sequence of one or more commands; sent from the controller to an instrument. Conversely, a Response Message is the data from the instrument to the controller.

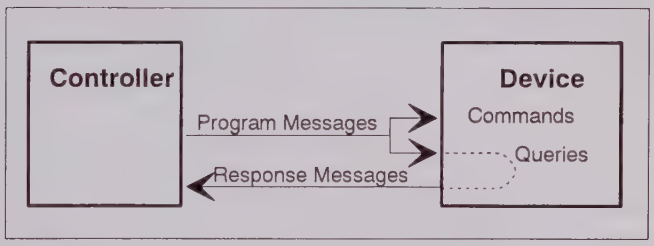


Fig. 3-4 *Program and response messages.*

The GPIB controller instructs the device through program messages. The device will only send responses when explicitly requested to do so; that is when the controller sends a query. Queries are recognized by the question mark at the end of the header, for example: *IDN? (requests the instrument to send identity data).

Syntax and Style

Syntax of Program Messages

A command or query is called a program message unit. A program message unit consists of a header followed by one or more parameters, as shown in the following syntax diagram.

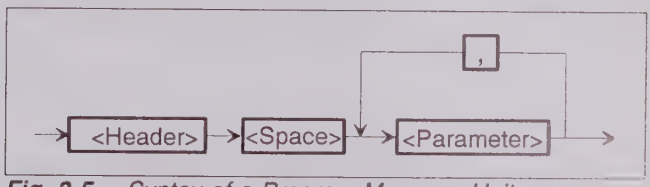


Fig. 3-5 *Syntax of a Program Message Unit.*

One or more program message units (commands) may be sent within a simple program message, see fig.3- 6

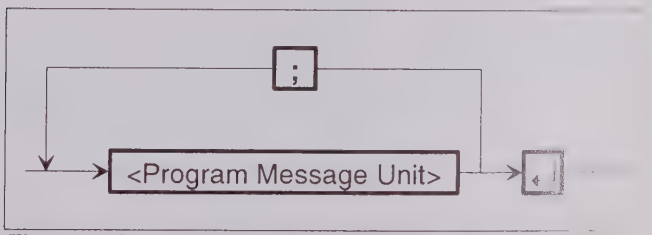


Fig 3-6 *Syntax of a terminated Program Message*

The ↵ is the pmt (program message terminator) and it must be one of the following codes:

↵	NL^END	This is <new line> code sent concurrently with the END message on the GPIB.
	NL	This is the <new line> code.
	<dab>^END	This is the END message sent concurrently with the last data byte <dab>.

NOTE: NL is the same as the ASCII LF (<line feed> = ASCII $10_{decimal}$).
The END message is sent via the EOI-line of the GPIB.
The ^ character stands for 'at the same time as'.

Most controller programming languages send these terminators automatically, but allow changing it. So make sure that the terminator is as above.

Example of a terminated program message:

program message unit program message unit terminator

This program message consists of two message units. The unit separator (semicolon) separates message units.

Basically there are two types of commands:

Common commands

The common command header starts with the asterisk character (*), for example *RST.

SCPI commands

SCPI command headers may consist of several keywords (mnemonics), separated by the colon character (:).

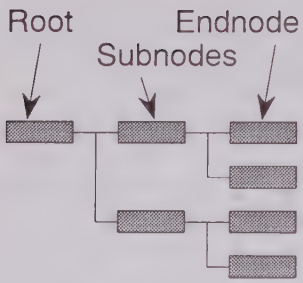


Fig. 3-7 The SCPI command tree.

Each keyword in a SCPI command header represents a node in the SCPI command tree. The leftmost keyword (INPut in the previous example) is the root level keyword, representing the highest hierarchical level in the command tree.

The keywords following represent subnodes under the root node. See 'COMMAND TREE' on page 3-16 for more details of this subject.

● *Forgiving Listening* The syntax specification of a command is as follows:

ACQusition:APERture␣<numeric value>

Where: ACQ and APER specify the short form, and ACQusition and APERture specify the long form. However, ACQU or APERT are not allowed and cause a command error.

In program messages either the long or the short form may be used in upper or lower case letters. You may even mix upper and lower case. There is no semantic difference between upper and lower case in program messages. This instrument behavior is called forgiving listening.

For example, an application program may send the following characters over the bus:

SEND→ **`iNp:AtT 1`**

short form used in a mix of upper and lower case

SEND→ **`Input:Attenuation 1`**

long and short form used in a mix of upper and lower case.

● *Notation habit in command syntax* To clarify the difference between short and long form, the short form in a syntax specification is shown in upper case letters and the remaining part of the long form in lower case letters.

Notice however, that this does not specify the use of upper and lower case characters in the message that you actually sent. Upper and lower case letters, as used in syntax specifications, are only a notation convention to ease the distinction between long and short form.

Syntax of Response Messages

The response of a SCPI instrument to a query (response message unit) consists of one or more parameters (data elements), as the following syntax diagram shows. There is no header returned.

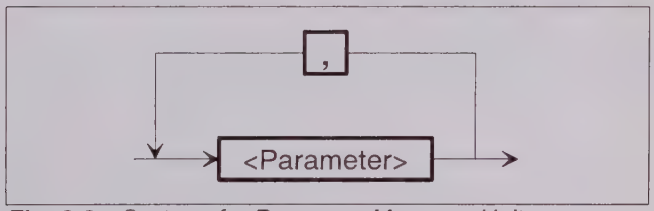


Fig. 3-8 *Syntax of a Response Message Unit.*

If there are multiple queries in a program message, the instrument groups the multiple response message units together in one response message according to the following syntax:

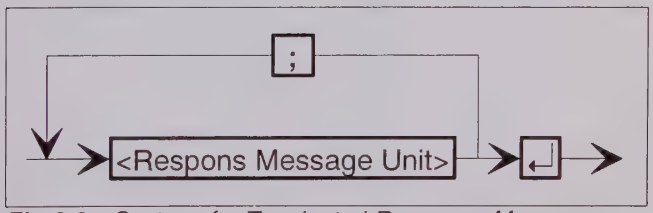


Fig 3-9 *Syntax of a Terminated Response Message.*

The response message terminator (rmt) is always NL^END, where:

NL^END is <new line> code (equal to <line feed> code = ASCII 10 decimal) sent concurrently with the END message. The END message is sent by asserting the EOI line of the GPIB bus.

- *Responses:* A SCPI instrument always send its response data in short form and in capitals.

Example:

You program an instrument with the following command:

SEND→ `:ROSCillator:SOURce EXTernal`

Then you send the following query to the instrument:

SEND→ `:ROSCillator:SOURce?`

The instrument will return:

READ← `EXT`

response in short form and in capitals

Command Tree

Command Trees like the one below are used to document the SCPI command set in this manual. The keyword (mnemonic) on the root level of the command tree is the name of the subsystem. The following example illustrates the Command Tree of the INPut1 subsystem.

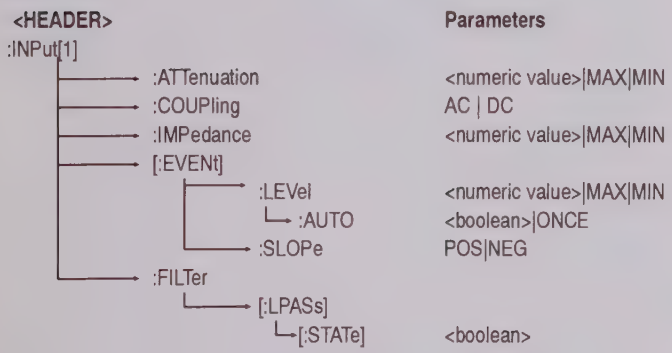


Fig. 3-10 *Example of an INPut subsystem command tree.*

NOTE: The keywords placed in square brackets are so-called optional nodes. This means that you may omit them from the program message.

Example:

SEND→ **`INPUT1:FILTER:LPASS:STATE␣ON`**

is the same as

SEND→ **`INPUT:FILTER␣ON`**

Moving down the Command Tree

The command tree shows the paths you should use for the command syntax. A single command header begins from the root level downward to the 'leaf nodes' of the command tree. (Leaf nodes are the last keywords in the command header, before the parameters.)

● *Example:* SEND→ **`INPut:FILTer:LPAS:STATe`**

Where: INPut is the root node and STATe the leaf node.

Each colon in the command header moves the current path down one level from the root in the command tree. Once you reach the leaf node level in the tree, you can add several leaf nodes without having to repeat the path from the root level.

Just follow the rules below:

- Always give the full header path, from the root, for the first command in a new program message.
- For the following commands within the same program message, omit the header path and send only the leaf node (without colon).

NOTE: You can only do this if the header path of the new leaf-node is the same as that of the previous one. If not, the full header path must be given starting with a colon.

Command header = Header path + leaf node

- Once you send the pmt (program message terminator), the first command in a new program message must start from the root.

● *Example:* SEND→ **`INPut:EVENt:LEVel`**

This is the command header where:
INPut:EVENt is the header path and
:LEVel is the leaf-node.

This means, INPut:EVENt:LEVel 0.5;SLOPe NEG is a valid program message, because SLOPe is also a leaf-node under the header path INPut:EVENt.

NOTE: There is no colon before SLOPe!

Numeric Data

Decimal data are printed as numerical values throughout this manual. Numeric values may contain both a decimal point and an exponent (base 10).

These numerals are often represented as NRf (NR = NumeRic, f = flexible) format.

● *Keywords*

Besides entering decimal data as numeric values, several keywords can exist as special forms of numeric data, such as MINimum, MAXimum, DEFault, STEP, UP, DOWN, NAN (Not A Number), INFinity, NINF (Negative INFinity). The Command Reference chapters explicitly specify which keywords a particular command allow.

DEFault

This keyword means that the value of the current parameter will be set to default if you do not specify a value. The device can take a specific value or may leave the setting unchanged, dependent upon the device's choice.

MINimum

This keyword sets a parameter to its minimum value.

MAXimum

This keyword sets a parameter to its maximum value.

The instrument always allows MINimum and MAXimum as a data element in commands, where the parameter is a numeric value. MIN and MAX values of a parameter can always be queried.

Example:

SEND→ `INP:ATT? MAXimum`

This query returns the maximum range value.

● *Suffixes*

You can use suffixes to express a unit or multiplier that is associated with the decimal numeric data. Valid suffixes are s (seconds), ms (milliseconds), mohm (Mega Ohm), kHz (kilo Herts), mV (millivolt).

Example:

SEND→ `SENSe:ACQusition:APERture 100ms`

Where: ms is the suffix for the numeric value 100.

Notice that you may also send ms as MS or mS. MS does still mean milliseconds, not Mega Siemens!

Response messages do not have suffixes. The returned value is always sent using standard units, as V, S, Hz, unless you explicitly specify a default unit by a FORMat command.

Booleans

A Boolean parameter specifies a single binary condition which is either true or false.

Boolean parameters

These can be one of the following:

ON or 1 means condition true.
OFF or 0 means condition false.

● *Example* SEND→ `:SYSTem:TOUT ON` or `SYSTem:TOUT 1`

This switches timeout monitoring on.

A query, for instance SYSTem:TOUT?, will return 1 or 0 never ON or OFF.

Expression Data

You must enclose expression program data in parentnesis (). Three possibilities of expression data are:

- <numeric expression data>
- <parameter list>
- <channel list>

An example of <numeric expression data> is:
(((1*X)–10.7E6)/1) This subtracts a 10.7 MHz intermediate frequency from the measured result.

An example of <parameter list> is:
(5,0.02) This is a list of two parameters, the first one is 5 and the second one 0.02.

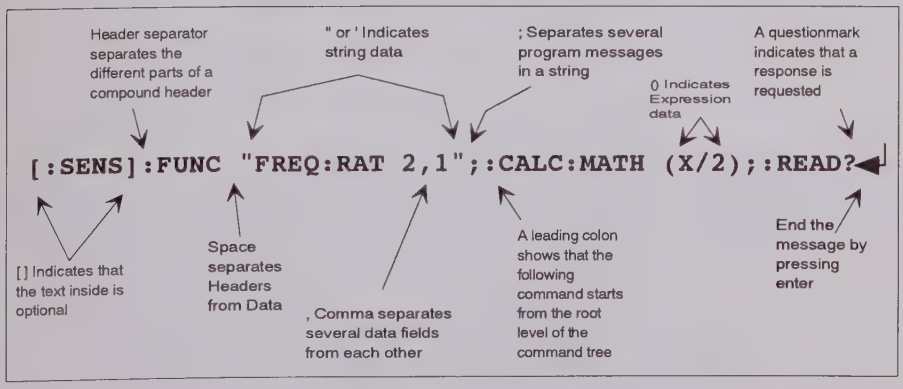
An example of <channel list> is:
(@2),(@1) This specifies channel 2 as the main channel and channel 1 as the second channel.

Other data types

Other data types that can be used for parameters are:

- String data: Always enclosed between single or double quotes, for example "This is a string" or 'This is a string.'
- Character data: For this data type, the same rules apply as for the command header mnemonics. For example: POSitive, NEGative, EITHer.
- Non-decimal data: For instance, #H3A for hexadecimal data.
- Block data: Used to transfer any 8-bit coded data. This data starts with a preamble that contains information about the length of the parameter. Example:
  #218INP:IMP_50;COUP_DC

Summary


Header separator separates the different parts of a compound header
" or ' Indicates string data
; Separates several program messages in a string
0 Indicates Expression data
A questionmark indicates that a response is requested
[:SENS]:FUNC "FREQ:RAT 2,1";:CALC:MATH (X/2);:READ?
[] Indicates that the text inside is optional
Space separates Headers from Data
, Comma separates several data fields from each other
A leading colon shows that the following command starts from the root level of the command tree
End the message by pressing enter

Status Reporting System

Structure of the Status System

The instruments uses the status structure to reflect the current instrument condition and to report all kind of events that happened. It ultimately reports the status in its Status Byte Register.

The status structure consists of the Register Model and Queue Model structures. The Standard Event Status is an example of a Register Model structure; the Output Queue is an example of a Queue Model structure. The Status Byte Register summarizes all Status Models, registers, and queues. From the Status Byte Register it is always possible to trace back to the event that set the corresponding summary bit. The PM 6680 status system is described in chapter 14, 'Status Subsystem' .

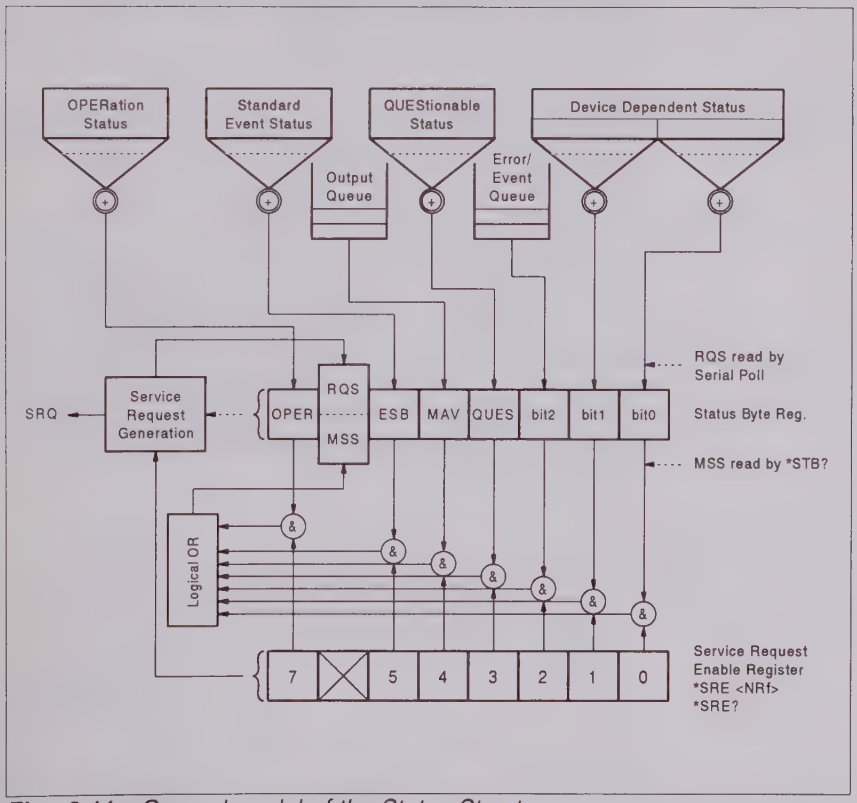


Fig. 3-11 *General model of the Status Structure.*

Error Reporting

An instrument will place detected errors in its Error Queue. This queue is a FIFO (First In First Out) buffer. When you read the queue, the first error will come out first, the last error last. If the queue overflows, an overflow message is placed last in the queue, and further errors are thrown away until there is room in the queue again.

Bit 2 in the Status Byte Register shows if the instrument has detected errors. It is also possible to enable this bit for Service Request on the GPIB. This can then interrupt the GPIB controller program when an error occur.

This is further described in chapter 19, 'Error Messages'.

Initialization and resetting

Reset strategy

There are three levels of initialization.

- Bus initialization
- Message exchange initialization
- Device initialization

● *Bus initialization*

This is the first level of initialization. The controller program should start with this which initializes the IEEE-interfaces of all connected instruments. It puts the complete system into remote enable (REN-line active) and the controller sends the interface clear (IFC) command. The command or the command sequence for this initialization is controller and language dependent. Refer to the user manual of the system controller in use.

● *Message exchange initialization*

Device clear is the second level of initialization. It initializes the bus message exchange, but does not affect the device functions. Device clear can be signalled either with DCL (to all instruments) or SDC (Selective device clear) only to the addressed instruments. The instrument action on receiving DCL and SDC is identical. See 'Device clear' on page 3-7.

When you use DCL, remember that it is always sent to all connected instruments and that old instruments (pre 488.2) may for instance set default settings when receiving this command.

● *Device initialization*

The third level of initialization is on the device level. This means that it concerns only the addressed instruments.

The ✱RST command

Use this command to reset a device. It initializes the device-specific functions in the PM 6680.

The following happens when you use the ✱RST command:

- You set the Timer/Counter-specific functions to a known default state. The ✱RST condition for each command is given in the command reference chapters.

- You set the macro, defined by *DDT to a device-defined state. (See *DDT in chapter 18, 'Trigger Subsystem'.
- You disable macros.
- You set the timer/counter in an idle state (outputs are disabled), so that it can start new operations.

The *CLS command

Use this command to clear the status data structures. See chapter 14, 'Status Subsystem'.

The following happens when you use the *CLS command:

- The instrument clears all event registers summarized in the status byte register.
- It empties all queues, which are summarized in the status byte register, except the output queue, which is summarized in the MAV bit.

This page is intentionally left blank.

Chapter 4

Instrument Model

Introduction

The figure below shows how the PM 6680 instrument functions are categorized. This instrument model is fully compatible with the SCPI generalized instrument model.

The standardized SCPI instrument model, contains three major instrument categories as shown in the following table:

Function	Instrument type	Examples
Signal acquisition	Sense instruments	Voltmeter, oscilloscope, counter
Signal generation	Source instruments	Pulse generator, power supply
Signal routing	Switch instruments	Scanners, (de-) multiplexers

An instrument may use a combination of the above functions. The PM 6680 belongs to the Signal acquisition category, and only that category is described in this manual.

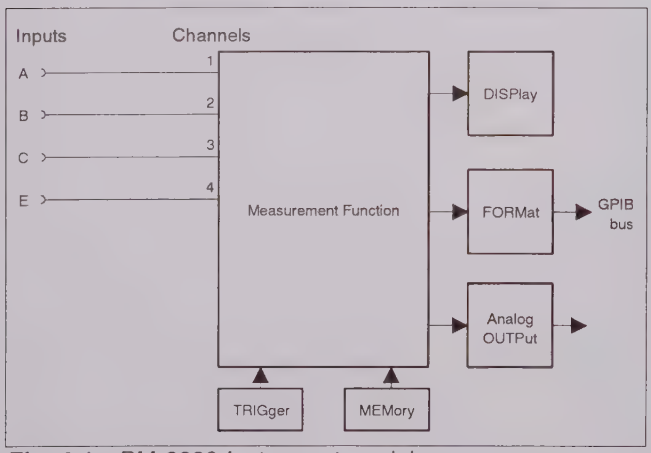


Fig. 4-1 *PM 6680 Instrument model.*

The instrument model in fig 4-1 defines where elements of the PM 6680 language are assigned in the command hierarchy. The major signal function areas are shown broken into blocks. Each of these blocks are major command sub-trees in the PM 6680 command language.

The instrument model also shows how measurement data and applied signals flow through the instrument. The model does not include the administrative data flow associated with queries, commands, performing calibrations etc.

Measurement Function Block

The measurement function block converts the input signals into an internal data format that is available for formatting into GPIB bus data. The measurement function is divided into three different blocks: INPut, SENSe and CALCulate. See fig. 4-2.

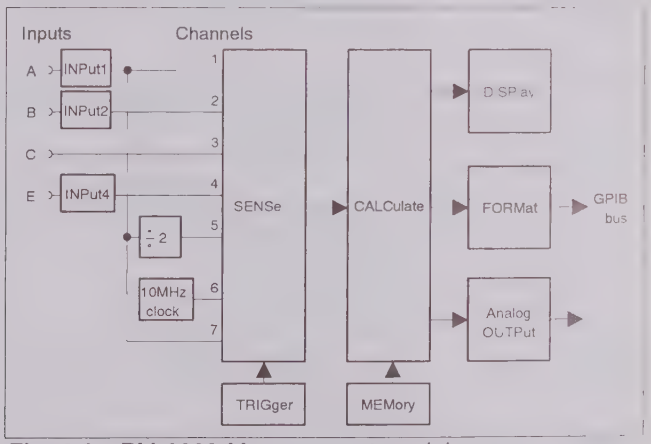


Fig. 4-2 PM 6680 Measurement model.

- *INPut*

The INPut block performs all the signal conditioning of the input signal before it is converted into data by the SENSe block. The INPut block includes coupling, impedance, filtering etc.

- *SENSe*

The SENSe block converts the signals into internal data that can be processed by the CALCulate block. The SENSe com-

mands control various characteristics of the measurement and acquisition process. These include: gate time, measurement function, resolution, etc.

- *CALCulate* — The CALCulate block performs all the necessary calculations to get the required data. These calculations include: calibration, statistics, mathematics, etc.

Other subsystems

In addition to the major functions (subsystems), there are several other subsystems in the instrument model.

The different blocks have the following functions.

- *Analog OUTPut* — This subsystem controls the analog output available in the PM 6680.
- *CALibration* — This subsystem controls the calibration of the interpolators used to increase the resolution of PM 6680.
- *DISPlay* — Commands in this subsystem control what data is to be present on the display, and wether the display is on or off.
- *FORMat* — The FORMat block converts the internal data representation to the data transferred over the external GPIB interface. Commands in this block control the data type to be sent over the external interface.
- *MEMory* — The MEMory block holds macro and instrument state data inside the PM 6680.
- *STATus* — This subsystem can be used to get information about what is happening in the instrument at the moment.
- *Synchronization* — This subsystem can be used to synchronize the measurements with the controller.
- *SYSTem* — This subsystem controls some system parameters like timeout.
- *TEST* — This subsystem tests the hardware and software of the counter and reports errors.

● *TRIGger*

The trigger block provides the PM 6680 with synchronization capability with external events. Commands in this block control the trigger and arming functions of the Timer/ Counter.

About all commands

- All commands in PM 6680 are sequential, i.e., they are executed in the same order as they are received.
- If a new measurement command is received when a measurement is already in progress, the measurement in progress will be aborted, unless *WAI is used before the command.

MEASurement function

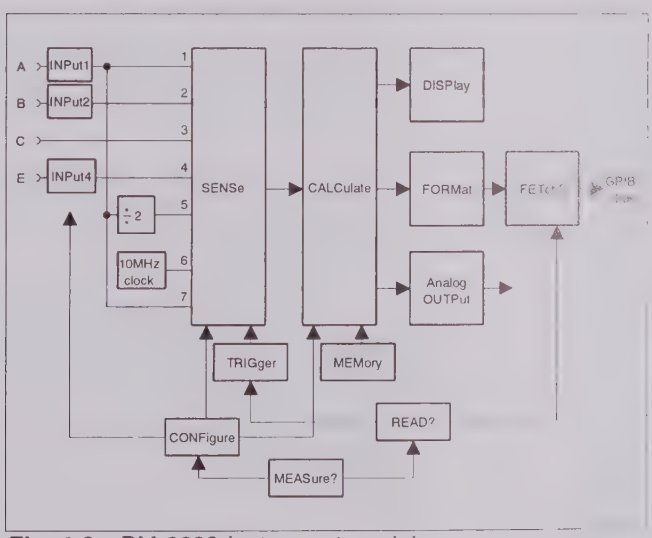


Fig. 4-3 *PM 6680 Instrument model*

In addition to the subsystems of the instrument model which controls the instrument functions, SCPI has signal-oriented functions to obtain measurement results. This group of MEASure functions has a different level of compatibility and flexibility. The parameters used with commands from the MEASure group describe the signal you are going to measure. This means that the MEASure? functions give

compatibility between instruments, since you don't need to know anything about the instrument you are using.

- *MEASure?*

This is the most simple command to use, but it does not offer much flexibility. The MEASure? command lets the counter configure itself for an optimal measurement, start the data acquisition, and return the result.

- *CONFigure; READ?*

The CONFigure command makes the counter choose an optimal setting for the specified measurement. CONFigure may cause any device setting to change. READ? starts the acquisition and returns the result.

This sequence does the same as the MEASure command, but now it is possible to insert commands between CONFigure and READ? to adjust the setting of a particular function (called fine tuning). For instance, you can set an input attenuator at a required value.

- *CONFigure; INITiate;FETCh?*

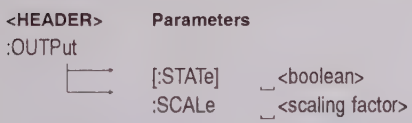
The READ? command can be divided into the INITiate command, which starts the measurement, and the FETCh? command, which requests the instrument to return the measuring results to the controller.

Versatility of measurement commands	
MEASure?	Simple to use, few additional possibilities.
CONFigure READ?	Somewhat more difficult, but some extra possibilities.
CONFigure INITiate FETCh?	Most difficult to use, but many extra features.

Chapter 5

Analog Output Subsystem

Analog Output Subsystem Command Tree



Enable Analog Out — command/query

This command switches on/off the analog output. See also OUTput:SCALe command on the next page.

Command syntax: **:OUTPut[:STATe]**␣<boolean>

Where: <boolean> = (1|ON / 0|OFF)

Query syntax: :OUTPut[:STATe]?

● *Returned format:* <1|0>↵

✷RST condition: OFF

Complies to standards: SCPI 1991.0 confirmed.

Scaling Factor, Analog Output

Scaling Factor, Analog Output — command/query

Sets the scaling factor for the analog output. The measurement result is scaled after math, if math is used.

Command syntax: **:OUTPut:SCALe** < Decimal data >

Where:
<Decimal data> is the scaling factor. The range is -10^{20} to $+10^{20}$.

● *Example 1:* If you want a specific resolution in your recording the formula is:

$$\textit{Scaling factor} = \frac{1}{250 * \textit{Desired resolution}}$$

If you want a resolution of 0.001,

$$\textit{Scaling factor} = \frac{1}{250 * 0.001} = 4$$

Send→ `:OUTP:SCAL 4`

● *Example 2:* If you want a full-scale output for a specific readout the formula is:

$$\textit{Scaling factor} = \frac{\textit{Full scale value}}{250}$$

If you want full scale output (4.98 V) for a reading of 0.00359,

$$\textit{Scaling factor} = \frac{0.00359}{250} = 0.0000143$$

Send→ `:OUTP:SCAL 143E-7`

Query syntax:? :OUTPut:SCALe?

● *Returned format:* < Decimal data>↵

***RST condition:** 1

Complies to standards: Philips Timer/Counter 1991.

How to use the Analog Output

The analog output is switched off as default. You switch it on or off and set the scaling factor in this subsystem.

As default, the analog output converts the three most significant digits of the fraction (to the right of the decimal point) to a voltage.

NOTE: The position of the decimal point used to determine which digits are converted, is not the same as on the display or on the GPIB output. You must first figure out where the decimal point would be if you had a number without exponent, *that is, the result in Hz, seconds or Volt.* Then you know that the three digits to the right of that decimal point are the converted digits.

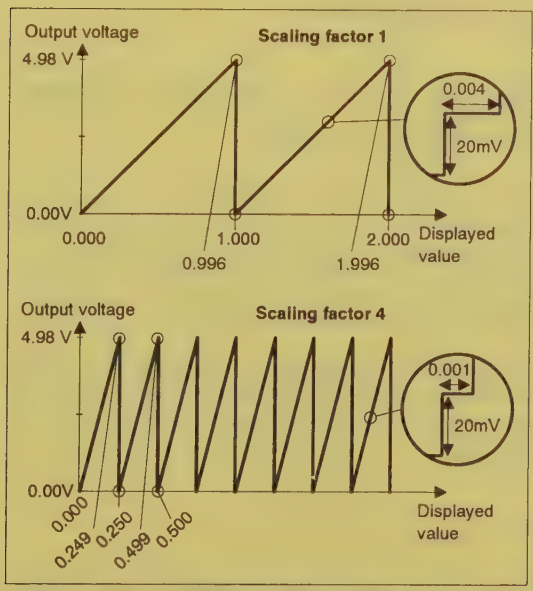


Fig. 5-1 *Output voltage versus displayed value for two different scaling factors.*

The analog output range is 0 to 4.98 V. This means that 0 V represents X.000 on the display and 4.98 V represents X.996 on the display. The minimum voltage change is a step of 20 mV. One such step is taken each time the display changes with X.004.

The X in the above paragraph can be any digit and does not influence the output voltage. If the display changes from 0.996 to 1.000 the voltage drops from 4.98 V to 0 V and if the display value increases further the output voltage starts to increase again.

If you wish to record other digits than the three to the right of the decimal point, you must append an exponent to the scaling factor. A negative exponent moves the three digit window to the left and a positive exponent moves the window to

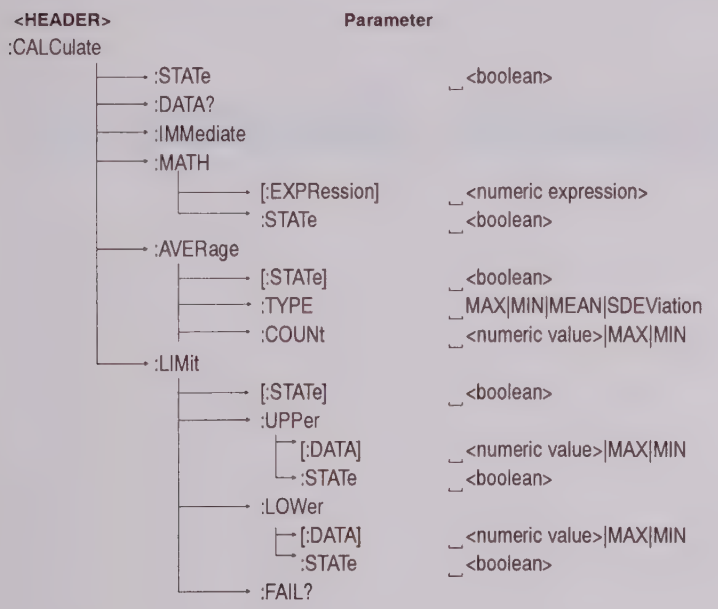
This page is intentionally left blank.

Chapter 6

Calculate Subsystem

Calculate Subsystem Command Tree

Calculate Subsystem Command Tree



Enable Calculation

command/query

Switches on/off the complete post processing block. If disabled then neither mathematics, averaging, statistics or limit monitoring can be done.

Command syntax: **: CALCulate:STATe** ␣<Boolean>

Where: <boolean> = (1|ON / 0|OFF)

Query syntax: :CALCulate:STATe?

● *Returned format:* 1|0↵

RST condition: OFF

Complies to standards: SCPI 1991.0, Confirmed

Read Calculated Data

Read Calculated Data — query

Reads out data calculated in the post processing block. Use this command to read out the result from a statistical functions. The normal READ?query will only output the last result without statistical calculations.

Query syntax: **:CALCulate:DATA?**

- *Returned format:* <decimal data>↵
- *Example:*

SEND→ `:CALC:AVER:STAT_ON;TYPE_SDEV;:INIT;*OPC`

Wait for operation complete

SEND→ `:CALC:DATA?`

READ← `<Value of standard deviation>`

****RST condition:*** Event, no RST condition.

Complies to standards: SCPI 1991.0, Confirmed

Recalculate Data — command/query

This event causes the calculate subsystem to reprocess the statistical function on the sense data without reacquiring the data. Query returns this re-processed data.

Command syntax: **:CALCulate:IMMediate**

Query syntax: :CALCulate:IMMediate?

- *Returned format:* <decimal data>↵

  Where: <decimal data> is the recalculated data.

- *Example:*

```
SEND→  :CALC:AVER:STAT ON;TYPE SDEV;:INIT;*OPC

Wait for operation complete

SEND→ :CALC:DATA?

READ← <Value of standard deviation>

SEND→ :CALC:AVER:TYPE MEAN

SEND→ :CALC:IMM?

READ← <Mean value>
```

***RST condition:** Event, no RST condition.

Complies to standards: SCPI 1991.0, Confirmed.

Select Mathematical Expression

Select Mathematical Expression **command/query**

Defines the mathematical expression used for mathematical operations. The expression could be one of the two expressions also available via the front panel.

NOTE: The data type for Expression is <expression data> which must be typed within parentheses.

Command syntax: **:CALCulate:MATH[:EXPRession]**␣(<expression>)

Where: <expression> is one of the following two mathematical expressions:

((K␣␣X)␣+␣L)␣/␣M or ((K␣/␣X)␣+␣L)␣/␣M*

K, L and *M* can be any positive or negative numerical constant, or *XOLD* for the last, previously measured value.

Each operand must be surrounded by space characters.

No deviations are allowed from these two forms.

● *Example* SEND→ `:CALC:MATH␣(((1␣*␣X)␣-␣0)␣/␣XOLD)`

This example gives a relative result from the last measuring result.

✷RST condition: ((1 ∗ *X*) + 0)/1 (No calculation)

Query syntax: :CALCulate:MATH[:EXPRession]?

● *Returned format:* <expression>↵

Where <expression> is the expression selected.

Complies to standards: SCPI 1991.0 Confirmed.

Enable Mathematics — command/query

Switches on/off the mathematical function. Note that the CALCulate subsystem is automatically enabled when MATH operations are switched on. This means that other enabled calculate sub-blocks are indirectly switched on.

Command syntax: **:CALCulate:MATH:STATe** _< Boolean >

Where: <boolean> = (1|ON / 0|OFF)

Query syntax: :CALCulate:MATH:STATe?

- *Returned syntax:* 0|1

***RST condition:** OFF

Complies to standards: SCPI 1991.0, confirmed.

Enable Statistics

Enable Statistics **command/query**

Switches on/off the statistical function. Note that the CALCulate subsystem is automatically enabled when the statistical functions are switched on. This means that other enabled calculate sub-blocks are indirectly switched on. The statistics must be enabled before the measurements are performed. When the statistical function is enabled, the counter will keep the trigger subsystem initiated until the :CALC:AVER:COUNT variable is reached. This is done without any change in the trigger subsystem settings. Consider that the trigger subsystem is programmed to perform 1000 measurements when initiated. In such a case, the counter must make 10000 measurements if the statistical function requires 9500 measurements because the number of measurements must be a multiple of the number of measurements programmed in trigger subsystem (1000 in this example)..

Command syntax: **:CALCulate:AVERage:STATe**_< Boolean >

Where: <boolean> = (1|ON / 0|OFF)

Query syntax: :CALCulate:AVERage:STATe?

● *Returned syntax:* 1|0

RST condition: OFF

Complies to standards: Philips Timer/Counter 1991.

Statistical Type — command/query

Selects the statistical function to be performed.

> **NOTE: You must use CALC:DATA? to read the result of statistical operations. READ?, FETC? will only send the results that the statistical operation is based on.**

Command syntax: **:CALCulate:AVERage:TYPE** MAX|MIN|MEAN|SDEViation

Where:

MAX returns the maximum value of all samples taken under CALC:AVER control.

MIN returns the minimum value of all samples taken under CALC:AVER control.

MEAN returns the mean value of the samples taken:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

SDEV Returns the standard deviation:

$$s = \sqrt{\frac{1}{n-1}\left(\sum X_i^2 - \frac{1}{n}\left(\sum X_i\right)^2\right)}$$

Query syntax: :CALCulate:AVERage:TYPE?

● *Returned format:* MAX|MIN|MEAN|SDEV ↵

***RST condition:** MEAN

Complies to standards: Philips Timer/Counter 1991.

Sample Size for Statistics

Sample Size for Statistics — command/query

Sets the number of samples to use in statistics sampling.

Parameter range: 1 to 65535.

Command syntax: **:CALCulate:AVERage:COUNt** ␣ < No. of samples>

Where <No. of samples> is a number in the range 1 to 65535.

Query syntax: :CALCulate:AVERage:COUNt?

● *Returned format:* < No. of samples>↵

****RST condition:*** 100

Complies to standards: Philips Timer/Counter 1991.

Enable Monitoring of Parameter Limits — command/query

Turns on/off the limit monitoring calculations.

Limit monitoring makes it is possible to get a service request when the measurement value falls below a lower limit or rises above an upper limit.

Two status byte bits are defined to support limit monitoring. One is set when the results are greater than the UPPer limit, the other is set when the result is less than the LOWer limit. The bits are enabled using the standard ✷SRE command and :STAT:DREG0:ENAB. Using both these bits, it is possible to get a service request when a value passes out of a band (UPPer is set at the upper band border and LOWer at the lower border) OR when a measurement value enters a band (LOWer set at the upper band border and UPPer set at the lower border).

Turning the limit monitoring calculations on/off will not influence the status register mask bits which determine whether or not a service request will be generated when a limit is reached. Note that the calculate subsystem is automatically enabled when limit monitoring is switched on. This means that other enabled calculate sub-blocks are indirectly switched on.

Command syntax: **:CALCulate:LIMit[:STATe]**␣<Boolean>
Where: <boolean> = (1|ON / 0|OFF)

Query syntax: :CALCulate:LIMit[:STATe]?

● *Returned format:* 1|0↵

✷RST condition: OFF

See also: Example two in chapter 20 deals with limit monitoring.

Complies to standards: SCPI 1991.0, confirmed.

Set Upper Limit

Set Upper Limit — command/query

Sets the value of the 'Upper Limit', i.e., the highest measurement result allowed before the counter generates a 1 that can be read with :CALCulate:LIMit:FAIL?, or by reading the corresponding status byte.

Parameter range: -10^{20} to $+10^{20}$

Command syntax: **:CALCulate:LIMit:UPPer[:DATA]** <Decimal data>|MAX|MIN

Query syntax: :CALCulate:LIMit:UPPer[:DATA]?

● *Returned format:* <Decimal data>↵

RST condition: 0

Complies to standards: SCPI 1991.0, confirmed.

Check Against Upper Limit

command/query

Selects if the measured value should be checked against the upper limit.

Command syntax: **:CALCulate:LIMit:UPPer:STATe** <Boolean>

Where: <boolean> = (1|ON / 0|OFF)

Query syntax: :CALCulate:LIMit:UPPer:STATe?

- *Returned format:* 1| 0 ↵

***RST condition:** 0

Complies to standards: SCPI 1991.0, confirmed.

Set Low Limit

Set Low Limit — command/query

Sets the value of the 'Lower Limit' , i.e., the lowest measurement result allowed before the counter generates a 1 that can be read with :CALCulate:LIMit:FAIL?, or by reading the corresponding status byte

Parameter range: -10^{20} to $+10^{20}$.

Command syntax: **:CALCulate:LIMit:LOWer[:DATA]** <Decimal data>|MAX|MIN

Query syntax: :CALCulate:LIMit:LOWer[:DATA]?

● *Returned format:* < Decimal data>↵

✷RST condition: 0

Complies to standards: SCPI 1991.0, confirmed.

Check Against Lower Limit — command/query

Selects if the measured value should be checked against the lower limit.

Command syntax: **:CALCulate:LIMit:LOWer:STATe** <Boolean>

Where: <boolean> = (1|ON / 0|OFF)

Query syntax: :CALCulate:LIMit:LOWer:STATe?

- *Returned format:* 1|0 ↵

***RST condition:** 0

Complies to standards: SCPI 1991.0 confirmed.

Limit Fail

Limit Fail — query

Returns a 1 if the limit testing has failed (the measurement result has passed the limit), and a 0 if not.

The following events reset the fail flag:

- Power ON
- *RST
- A :CALC:LIM:STAT:␣ OFF→:CALC:LIM:STAT:␣ ON transition
- Reading a 1 with this command.

Query syntax: **:CALCulate:LIMit:FAIL?**

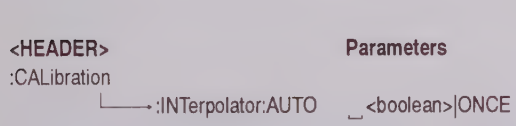
● *Returned format:* 1| 0 ↵

Complies to standards: SCPI 1991.0, confirmed.

Chapter 7

Calibration Subsystem

Calibration Subsystem Command Tree



Calibration of Interpolator — command/query

This reciprocal counter uses an interpolating technique to increase the resolution. In time measurements for example, interpolation increases the resolution from 100 ns to 0.5 ns.

The counter calibrates the interpolators automatically once for every measurement when this command is ON. When OFF the counter does no calibrations but uses the values from the last preceding calibration. When the command SENS:ACQ:RES_LOW is used, no calibrations are performed independent of the CAL:INT:AUTO command. The intention of this command is to switch off the auto calibration for applications that dump measurements into the internal memory. This will increase the measurement speed.

Command syntax: **:CALibration:INTerpolator:AUTO** _<boolean>| ONCE

Where: <Boolean> = (1 | ON / 0 | OFF)

Query syntax: :CALibration:INTerpolator:AUTO?

- *Returned format:* 1|0↵

RST condition: ON

See also: Chapter 19, 'How to Measure',.

Complies to standards: Philips Timer/Counter 1991.

This page is intentionally left blank.

Chapter 8

Display Subsystem

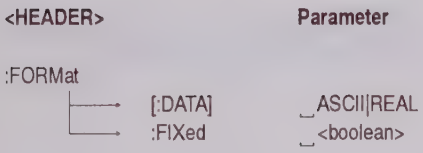
This page is intentionally left blank.

Chapter 9

Format Subsystem

Format Subsystem Command Tree

Format Subsystem Command Tree



Response Data Type — command/query

Sets the form in which format the result will be sent on the bus.

Command syntax: **:FORMat[:DATA]** ASCii|REAL

ASCII will send the measurement result in ASCII form.
<sign><mantissa value>E<sign><exponent value>
Where:
<sign> = + or –
<mantissa value> = 1 to 12 digits (depending on measuring resolution) plus one decimal point.
<exponent value> = 1 to 3 digits

REAL will send the result in binary IEEE Double Precision floating point format in a block-data element.
#18<8 bytes real>
The <8 bytes real> is a double precision binary floating point response according to IEEE488.2/IEEE754. This means that the eight bytes are sent in the following order:

First byte: <sign><7 MSB of the exponent>
Second byte: <4 LSB of the exponent><4 MSB of the fraction>
Third through eight byte: <48 LSB of the fraction>

Query syntax: :FORMat[:DATA]?

● *Returned format:* ASC|REAL

✷RST condition: ASCII

Complies to standards: SCPI 1991.0, confirmed.

Response Data Format

Response Data Format — command/query

Sets the ASCII format to fixed. This results in the following response format:

<sign><mantissa value>E<sign><exponent value>

Where:

<sign> = +|–
<mantissa value> = 12 digits plus one decimal point.
<exponent value> = 3 digits

The counter will add leading zeroes when the measurement resolution is less than 12 digits.

Command syntax: **:FORMat:FIXed**_<boolean>

Where: <Boolean> = (1 | ON / 0 | OFF)

Query syntax: :FORMat:FIXed?

● *Returned format:* 1|0 ↵

****RST condition:*** OFF

Complies to standards: Philips Timer/Counter 1991.

Chapter 10

Input Subsystems

Input Subsystems Command Tree

INPUT A

<HEADER>	Parameters
:INPut[1]	
:ATTenuation	_<numeric value>|MAX|MIN
:COUPling	_AC | DC
:IMPedance	_<numeric value>|MAX|MIN
[:EVENt]	
:LEVel	_<numeric value>|MAX|MIN
:AUTO	_<boolean>|ONCE
:SLOPe	_POS|NEG
:FILTer	
[:LPASs]	
[:STATe]	_<boolean>

INPUT B

<HEADER>	Parameters
:INPut2	
:ATTenuation	_<numeric value>|MAX|MIN
:COUPling	_AC | DC
:IMPedance	_<numeric value>|MAX|MIN
[:EVENt]	
:LEVel	_<numeric value>|MAX|MIN
:AUTO	_<boolean>|ONCE
:SLOPe	_POS|NEG
:COMMon	_<boolean>

INPUT E

<HEADER>	Parameters
:INPut4	
[:EVENt]	
:SLOPe	_POS|NEG

Attenuation

command/query

Attenuates the input signal with 1 or 10. The attenuation is automatically set if the input level is set to AUTO.

Command syntax:

- *Input A (1)* **:INPut[1]:ATTenuation**␣<Numeric value>|MAX| MIN
- *Input B (2)* **:INPut2:ATTenuation**␣<Numeric value>|MAX|MIN

Where:

<Numeric values> ≤ 5, and MIN gives attenuation 1.
<Numeric values> > 5, and MAX gives attenuation 10.

Query syntax:

- *Input A (1)* :INPut[1]:ATTenuation?
- *Input B (2)* :INPut2:ATTenuation?
- *Returned format:* 1.00000000000E+000|1.00000000000E+001 ↵

✷RST condition

- *Input A (1):* 1 (but set by autotrigger since autotrigger is on after ✷RST. (:INP:LEV:AUTO␣ON).
- *Input B (2):* 1 (but set by autotrigger since autotrigger is on after ✷RST. (:INP:LEV:AUTO␣ON).

Complies to standards: SCPI 1991.0, confirmed.

AC/DC Coupling

AC/DC Coupling — command/query

Selects AC coupling (normally used for frequency measurements), or DC coupling (normally used for time measurements).

Command syntax:

- *Input A (1)* **:INPut[1]:COUPling**_AC|DC
- *Input B (2)* **:INPut2:COUPling**_AC|DC

Query syntax:

- *Input A (1)* :INPut[1]:COUPling?
- *Input B (2)* :INPut2:COUPling?
- *Returned format:* AC|DC↵

✷RST condition

- *Input A (1):* AC
- *Input B (2):* DC

Complies to standards: SCPI 1991.0, confirmed.

Input Impedance — command/query

The impedance can be set to 50 Ω or 1 MΩ.

Command syntax:

- *Input A* — **:INPut[1]:IMPedance**␣<decimal data>|MAX|MIN
- *Input B* — **:INPut2:IMPedance**␣<decimal data>|MAX|MIN

If MIN or <decimal data> rounds off to 50 or less, it sets the input impedance to 50 Ω, and if MAX or <decimal data> rounds off to 1001 or more it sets the impedance to 1 MΩ.

Query syntax:

- *Input A* — :INPut[1]:IMPedance?
- *Input B* — :INPut2:IMPedance?
- *Returned format:* — 5.00000000000E+001|1.00000000000E+6␤

✷RST condition

- *Input A (1):* — 1 MΩ
- *Input B (2):* — 1 MΩ

Complies to standards: SCPI 1991.0, confirmed.

Fixed Trigger Level

Fixed Trigger Level — command/query

Input A and input B can be individually set to autotrigger or to fixed trigger levels of between –5.1 V and +5.1 V in steps of 0.02 V. If the attenuator is set to 10X, the range is –51 V and +51 V in 0.2 V steps.

For autotrigger, see the following page.

Command syntax:

- *Input A* **:INPut[1][:EVENt]:LEVel** _ <decimal data>|MAX|MIN
- *Input B* **:INPut2[:EVENt]:LEVel** _ <decimal data>|MAX|MIN

Where:
<decimal data> is a number between –5.1 V and +5.1 V if att=1X and between –51 V and +51 V if att=10X.
MAX gives +51.0 V
MIN gives –51.0 V

Note that when using MAX and MIN as data, the counter always tries to set the trigger level to +51 V and –51 V. If the attenuator is set to 1X, it is impossible to set this trigger level and the counter will return an error message.

Query syntax:

- *Input A:* :INPut[1][:EVENt]:LEVel?
- *Input B:* :INPut2[:EVENt]:LEVel?
- *Returned format:* <decimal data>

***RST condition** 0 (but controlled by Autotrigger since AUTO is on after *RST)

Complies to standards: Philips Timer/Counter 1991.

Autotrigger — command/query

If set to AUTO, the counter automatically controls both the trigger level and the attenuation[1]. If you have a stable amplitude, use the :AUTO_ONCE, and the autotrigger will determine the trigger level once and then set a fixed level.

NOTE: From the bus, input A and input B are always set to autotrigger individually.

Command syntax:

- *Input A* **:INPut[1][:EVENt]:LEVel:AUTO**_<boolean>|ONCE
- *Input B* **:INPut2[:EVENt]:LEVel:AUTO**_<boolean>|ONCE

Where:

<boolean> = (1|ON / 0|OFF)

ONCE means that the autotrigger switches on, checks the signal, stores the trigger levels as manually set levels, and then switches off auto. This improves measuring speed.

Query syntax:

- *Input A* :INPut[1][:EVENt]:LEVel:AUTO?
- *Input B* :INPut2[:EVENt]:LEVel:AUTO?
- *Returned format:* 1|0↵

***RST condition** ON

Complies to standards: Philips Timer/Counter 1991.

1 The autotrigger function normally sets the trigger levels to 50 % of the signal amplitude. Two exceptions exists however:
Rise/Fall time measurements: Here the input 1 (A) trigger level is set to 10% and the Input 2 (B) trigger level is set to 90% of the amplitude.
Variable Hysteresis mode (channel 7): The input 1 (A) trigger level is set to 75% and the Input 2 (B) trigger level is set to 25% of the amplitude

Trigger Slope

Trigger Slope — command/query

Selects if the counter should trigger on a positive or a negative transition.

Command syntax:

- *Input A* **:INPut[1][:EVENt]:SLOPe** ␣POS|NEG
- *Input B* **:INPut2[:EVENt]:SLOPe** ␣<POS|NEG
- *Input E* **:INPut4[:EVENt]:SLOPe** ␣POS|NEG

Query syntax:

- *Input A* :INPut[1][:EVENt]:SLOPe?
- *Input B* :INPut2[:EVENt]:SLOPe?
- *Input E* :INPut4[:EVENt]:SLOPe?
- *Returned format:* POS | NEG ↵

***RST condition** POS

Complies to standards: SCPI 1991.0, confirmed.

Low Pass Filter — command/query

Switches on or off the low pass filter on input 1 (A). It has a cutoff frequency of 100 kHz.

Command syntax: **:INPut[1]:FILTer** _ <boolean>

Where <boolean> is 1/ON|0/OFF

Query syntax: :INPut[1]:FILTer?

- *Returned format:* 1|0↵

✷RST condition OFF

Complies to standards: SCPI 1991.0, confirmed.

Common via A

Common via A — command/query

The signal on input 1 (A) is connected internally to input 2 (B) also. The interconnection is made before the filter on Input 1 (A).

Command syntax: **:INPut2:COMMon** <Boolean>

Where: <Boolean > = 1/ON | 0/OFF

Query syntax: :INPUt2:COMMon?

● *Returned format:* 1|0 ↵

✷RST condition OFF

Complies to standards: Philips Timer/Counter 1991

Using the Input settings

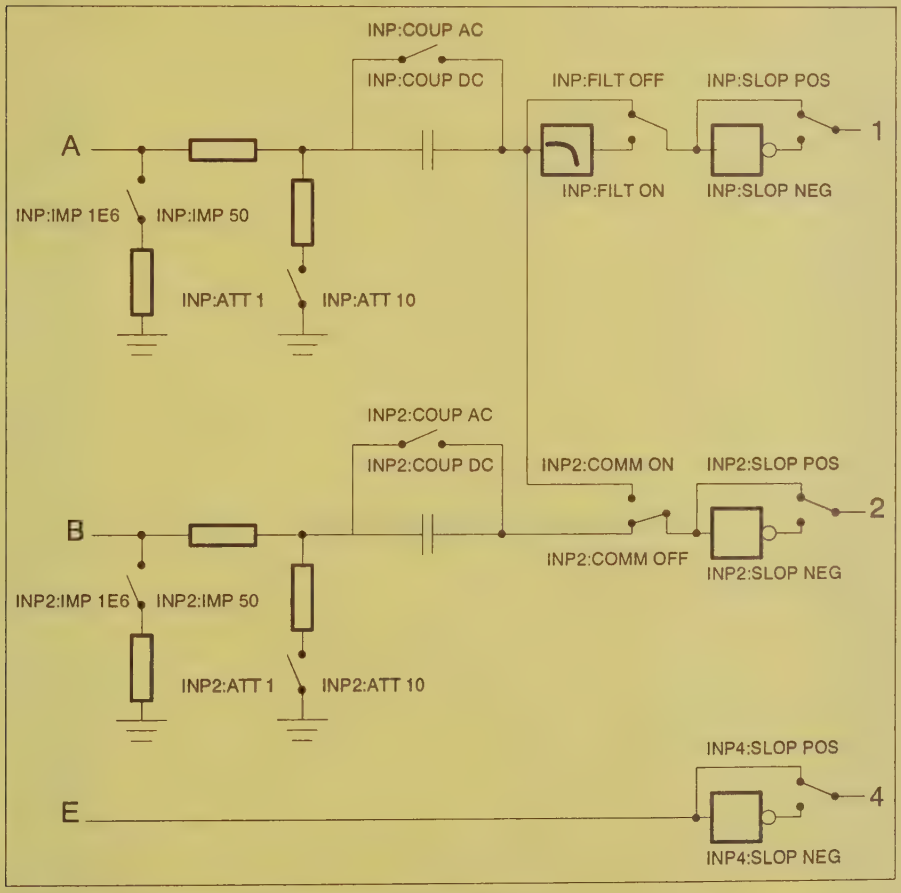


Fig. 10-1 *Summary of input amplifier settings.*

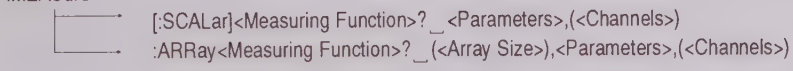
This page is intentionally left blank.

Chapter 11

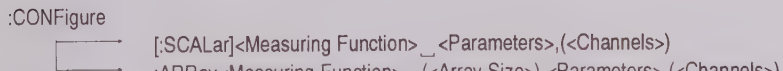
Measurement Function

Measure Function Command Trees

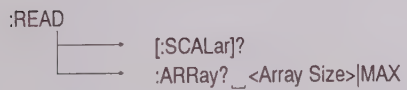
Set up the Instrument, Perform Measurement and Read Data



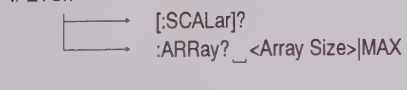
Set up Instrument for Measurement



Perform Measurement and Read Data



Fetch Measurement Data



Data to :MEASure and :CONFigure

<Measuring Function>	<Parameters>	<Channels>
[:VOLTage]		
:FREQuency		
[:CW]	[_[<expected value>[,<resolution>]]	[,(@1|@2|@3|@4|@5|@6|@7|)]]
:RATio	[_[<expected value>[,<resolution>]]	[,(@1|@2|@3|@4),(@1|@2|@3|@4)]0]
:BURSt	[_[<expected value>[,<resolution>]]	[,(@1@|2@|3@|4)]]
:PRFrequency	[_[<expected value>[,<resolution>]]	[,(@1|@2|@3|@4)]]
:PERiod	[_[<expected value>[,<resolution>]]	[,(@1|@2|@3|@4)]]
:TINTerval		[_,(@1|@2|@4),(@1|@2|@4)]
:PHASe	[_[<expected value>[,<resolution>]]	[,(@1|2),(@1|2)]]
:NWIDth	[_[<threshold>	[,(@1|@2|@4)]]
:PWIDth	[_[<threshold>	[,(@1|@2|@4)]]
:PDUTycycle|DCYCle	[_[<threshold>]	[,(@1|@2|@4)]]
:NDUTycycle	[_[<threshold>]	[,(@1|@2|@4)]]
:RISE:TIME	[_[<lower threshold>[,<upper threshold>[,<expected value>[,<resolution>]]]]	[,(@1)]]
:FALL:TIME	[_[<lower threshold>[,<upper threshold>[,<expected value>[,<resolution>]]]]	[,(@1)]]
:MAXimum		[_(@1|@2)]]
:MINimum		[_(@1|@2)]]
:PTPeak		[_(@1|@2)]]
:TOTalize		
:GATed		[,(@1|@2|@4)[,(@1|@2|@4)]]]
:ACCumulated	[_[<time for gate open>]	[,(@1|@2|@4)[,(@1|@2|@4)]]]
:SSTop		[_(@1|@2|@4)[,(@1|@2|@4)]]]
:TIMed	[_[<time for gate open>]	[,(@1|@2|@4)[,(@1|@2|@4)]]]
:CONTinuous*		[_(@1|@2|@4)[,(@1|@2|@4)]]]

* Only for :CONFigure

NOTE: The array size for :MEASure and :CONFigure and the channels and are expression data and they must be in parentheses ( ).

Measure once

Measure once — query

The measure query makes a complete measurement, including configuration and readout of data. Use measure when you can accept the generic measurement without fine tuning.

You cannot use the MEASure command for the :COUNt:CONTinuous function since this function measure without stopping (continuously for ever).

The MEASure command is a compound command identical to:
:ABORt;:CONFigure:<Meas_func>␣<Parameters, (<channels) ;:READ?

Query syntax: **:MEASure[:SCALar]:<Measuring Function>?**
[␣<parameters>[,(<channels>)]]
Where:
<Measuring Function>, <Parameters> and <Channels> are defined on page 11-3.
You may omit <parameters> and <Channels>, which are then set to default.

● *Returned format:* <data>↵
Where:
The format of the returned data is determined by the format commands: FORMat[:DATA] and FORMat:FIXed.

● *Example:* SEND→ `:MEAS:FREQ?`

READ← `1.78112526833E+004`

Type of command: Aborts all previous measurement commands if ✱WAI is not used.

See also: 'Explanations of the Measuring Functions' starting on page 11-13.

Complies to standards: SCPI 1991.0, confirmed.

Measure Array

query

The :MEASure:ARRay query differs from the :MEASure query in that it performs the number of measurements you decide in the <array size> and sends all the measuring results in one string to the controller.

The MEASure:ARRay command is a compound command identical to:
:ABORt;:CONFigure:ARRay:<Meas_func>␣(<array size),
<parameters>,(<channels>);:READ:ARRay?␣(<array size>)

Query syntax: **MEASure:ARRay:<Measuring Function>?**
␣(<array size>)[,<parameters>[,(<channels>)]]
Where:
<array size> sets the number of measurements in the array.

- *Returned format:* <Measuring result>{[,<measuring result>]}↵
- *Example:*

SEND→ :MEAS:ARR:FREQ?␣(10)
Ten measuring results will be returned.

Type of command: Aborts all previous measurement commands if not ✱WAI is used, see page 11-4.

Complies to standards: SCPI 1991.0, confirmed.

Configure — command/query

Use the configure command to set up the timer/counter to perform the measurement specified by <Measuring Function> , <parameters> and <channels>.

The :CONFigure command controls the settings of the Input, Sense and Trigger subsystems in the counter in order to make the best possible measurement. It also switches off any calculations with CALC:STATE␣OFF.
:READ? or INITiate;:FETCh? will make the measurement and read the resulting measured value.

Since you may not know exactly what settings the counter has chosen to configure itself for the measurement, it is advised to send an *RST before doing other manual set up measurements.

Command syntax: **:CONFigure[:SCALar]:<Measuring Function>**
[␣<parameters>[,(<channels>)]]
Where:
:SCALar means that one measurement is to be done.
<Measuring Function>, <Parameters> and <Channels> are defined on page 11-3.

Query syntax: :CONFigure?

● *Returned format:* <string>↵
Where:
<string> contains the setup configured by the last preceding CONFigure or MEASure command. The response is a <String response data element>.

See also: 'Explanations of the Measuring Functions' starting on page 11-13.

Complies to standards: SCPI 1991.0, confirmed.

Configure Array command

The :CONFigure:ARRay command differs from the :CONFigure command in that it sets up the counter to perform the number of measurements you choose in the <array size>.

To perform the selected function you must trigger the counter with the :READ:ARRay? or INITiate;:FETCh:ARRay? queries.

Command syntax: **:CONFigure:ARRay:<Measuring Function>**␣ (<array size>) [,<parameters>[,(<channels>)]]

Where:

<array size> sets the number of measurements in the array (1 to 2500).

<Measuring Function>, <Parameters>, and <Channels> are defined on page 11-3.

● *Example:* SEND→ `:CONF:ARR:PER␣(7),5E-3,1E-6,(@2)`

This example sets up the counter to make seven period measurements. The expected result is 5 ms, and the required resolution is 1 μs. Input B is the measurement input.

To make the measurements and fetch the seven measurement results:

SEND→ `:READ:ARR?␣7`

READ← `5.23421E-3,5.12311E-3,5.87526E-3, 5.50345E-3,5.33901E-3,5.25501E-3, 5.03571E-3`

Complies to standards: SCPI 1991.0, confirmed.

Read one Result

Read one Result — query

The read function performs new measurements and reads out a measuring result without reprogramming the counter. Using the :READ? query in conjunction with the :CONFigure command gives you a measure capability where you can fine tune the measurement.

If the counter is set up to do an array of measurements, :READ? makes all the measurements in the array, stores the results in the output buffer, and fetches the first measuring result. Use FETCh? to fetch other measuring results from the output buffer.

The :READ? query is identical to :ABORt;:INITiate;:FETCh?

Query syntax: **:READ[:SCALar]?**

- *Returned format:* <data>↵

  The format of the returned data is determined by the format commands :FORMat[:DATA] and FORMat:FIXed.

- *Example:* SEND→ `:CONF:FREQ;:INP:FILT␣ON;:READ?`

  This example configures the counter to make a standard frequency measurement with the 100 kHz filter on. The counter is triggered, and data from the measurement are read out with the :READ? query.

Type of command: Aborts all previous measurement commands if ✱WAI is not used.

Complies to standards: SCPI 1991.0, confirmed.

Read Array

query

The :READ:ARRay? query differs from the :READ? query by reading out several results at once, after making the number of measurements previously set up by :CONFigure:ARRay or MEASure:ARRAy?.

The READ:ARRay? query is identical to:
:ABORt;:INITiate;:FETCh:ARRay?␣<array size for FETCh>

Query syntax: **:READ:ARRay?**␣<array size for FETCh>|MAX

Where :

<array size for FETCh> sets the number of measuring results in the array. This size must be equal or less than the number of measurements specified with :CONFigure. MAX means that all the results in the output buffer will be fetched.

NOTE: The <array size for FETCh> does not tell :READ to make that many measurements, only to fetch that many results. :CONF:ARR, :MEAS:ARR or :ARM:LAY1:COUN sets the number of measurements.

● *Returned format:* <data>[,<data>]↵

The format of the returned data is determined by the format commands :FORMat[:DATA] and FORMat:FIXed.

● *Example:* SEND→ `:ARM:COUN␣10;:READ:ARR?␣5`

This example configures the counter to make an array of 10 standard frequency measurements. The counter is triggered and data from the first five measurements are read out with the :READ command.

Type of command: Aborts all previous measurement commands if ✱WAI is not used.

Complies to standards: SCPI 1991.0, confirmed.

Fetch one Result — query

The fetch query retrieves one measuring result from the measurement result buffer of the counter without making new measurements. Fetch does not work unless a measurement has been made by the :INITiate, :MEASure?, or :READ? commands.

If the counter has made an array of measurements, :FETCh? fetches the first measuring results first. The second FETCh? fetches the second result and so on. When the last measuring result has been fetched, fetch starts over again with the first result.

The same measuring result can be fetched again and again, as long as the result is valid, i.e., until:

- ✱RST is received
- an :INITiate, :MEASure or :READ command is executed
- any reconfiguration is done
- an acquisition of a new reading is started

If the measuring result in the output buffer is invalid but a new data measurement has been started, the FETCh query completes when a new measuring result becomes valid. If no new measurement has been started, an error is returned.

Query syntax: **:FETCh[:SCALar]?**

Where the optional :SCALar means that one result is retrieved.

● *Returned format:* <data>↵

The format of the returned data is determined by the format commands :FORMat[:DATA] and FORMat:FIXed.

Complies to standards: SCPI 1991.0, confirmed.

Fetch an Array of Results — query

FETCh:ARRay? query differs from the FETCh? query by fetching several measuring results at once.

An array of measurements must first be made by the commands. :INITiate, :MEASure:ARRay? or :CONFigure:ARRay; :READ?

If the array size is set to a positive value, the first measurement made is the first result to be fetched.

When the counter has made an array of measurements, :FETCh:ARRay?␣10 fetches the first 10 measuring results from the output queue. The second FETCh:ARRay?␣10 fetches the result 11 to 20 and so on. When the last measuring result has been fetched, fetch:array starts over again with the first result.

In totalizing for instance, you may want to read the last measurement result instead of the first one. This is possible if you set the array size to a negative number. Example: FETCh:ARRay?␣–5 fetches the last five results. The output queue pointer is not altered when the array size is negative. That is, the example above always gives the last five results every time the command is sent.

FETCh:ARRay?␣–1 is useful to fetch intermediate results in free running or array measurements, without interrupting the measurement.

Query syntax: **:FETCh:ARRay?**␣<fetch array size>|MAX

Where:

:ARRay means that an array of retrievals are done for each :FETCh command.

<fetch array size> is the number of retrievals in the array. This number must not exceed the number of measuring results in the measurement result buffer.

MAX means that all the results in the output buffer will be fetched.

Fetch an Array of Results

- *Returned format:* <data>[,<data>]↵
  The format of the returned data is determined by the format commands :FORMat[:DATA] and FORMat:FIXed.

- *Example:* If **MEAS:ARR:FREQ?␣(4)** gives the results
  1.1000,1.2000,1.3000,1.4000
  :FETC:ARR&space[-]**2** fetches the results
  1.1000,1.2000
  :FETC:ARR␣2 once more fetches the results
  1.3000,1.4000
  :FETC:ARR␣−1 always fetches the last result
  1.4000

Complies to standards: SCPI 1991.0, confirmed.

Explanations of the Measuring Functions

This sub-chapter explains the various measurements that can be done with :MEASure and :CONFigure;:READ. Only the queries for single measurements using the measure command are given here, but all of the information is also valid for the :CONFigure command and for both scalar (single) and array measurements.

Frequency — query

Traditional frequency measurements. The counter uses the <expected value> and <resolution> to calculate the measuring time (ACQuisition:APERture).

Query syntax: **:MEASure[:SCALar]:FREQuency[:CW]?**[␣ [<expected value>[,<resolution>]][,<(@1|2|3|4|5|6|7)>]]

Where the optional:

<expected value> is the expected frequency,
<resolution> is the required resolution.
<(@1|2|3|4|5|6|7)> is the channel to measure on:
(@1) means input A prescaled by 2[1]
(@2) means input B
(@3) means input C (HF-input option)
(@4) means input E (Rear panel arming input)
(@5) means input A prescaled by 2
(@6) means the internal 10 MHz reference
(@7) means input A with adjustable sensitivity

If you omit the channel, the instrument measures on input A.

● *Example:* SEND→ `:MEAS:FREQ?␣(@2)`

READ← `1.78112526833E+004`

This example measures the frequency at input B.

NOTE: The channel is expression data and it must be in parentheses ( ).

Complies to standards: SCPI 1991.0, confirmed.

1 The A input is always prescaled by 2 when measuring Frequency A.

Frequency Ratio — query

Frequency ratio measurements between two inputs.

Query syntax: **:MEASure[:SCALar]:FREQuency[:CW] :RATio?**
[_[<expected value>[,<resolution>]][,<(@1|2|3|4|5|6)>,<(@1|2|3|4|5|6)>]]

Where the optional:

<expected value> and <resolution> are ignored

<(@1|2|3|4|5|6)>,<(@1|2|3|4|5|6)> is the channels to measure on:

(@1) means input A
(@2) means input B
(@3) means input C (HF-input option)
(@4) means input E (Rear panel arming input)
(@5) means input A prescaled by 2
(@6) means the internal 10 MHz reference

If you omit the channel, the instrument measures between input A and input B.

● *Example:* SEND→ `:MEAS:FREQ:RAT?_(@1),(@3)`

READ← `2.345625764333E+000`

This example measures the ratio between input A and input C.

NOTE: The channel is expression data and it must be in parentheses ( ).

Complies to standards: SCPI 1991.0, confirmed.

Burst Carrier Frequency

Burst Carrier Frequency — query

Measures the carrier frequency of a burst. Use the SENS:ACQ:HOFF:TIME command to fine tune the burst synchronization time. This time should be longer than the burst, but shorter than the burst spacing. This implies that the burst duration is less than 50% of the pulse repetition frequency (PRF).

How to measure bursts is described in detail in the chapter 7 of the Operators Manual.

Query syntax: **:MEASure[:SCALar]:FREQuency:BURSt:CARRier?**
[_[<expected value>[,<resolution>]][,<(@1|2|3|4|6| 7)>]]

Where the optional:

<expected value> is the expected carrier frequency,
<resolution> is the required resolution, e.g. 1 gives 1 Hz resolution.
<(@1|2|3|4|5|6|7)> is the channel to measure on:
(@1) means input A
(@2) means input B
(@3) means input C (HF-input option)
(@4) means input E (Rear panel arming input)
(@5) means input A prescaled by 2
(@6) means the internal 10 MHz reference
(@7) means input A with adjustable sensitivity

If you omit the channel, the instrument measures on input A.

The <expected value> and <resolution> are used to calculate the measuring time (ACQuisition:APERture).

Complies to standards: Philips Timer/Counter 1991.

Pulse Repetition Frequency — query

Measures the PRF (Pulse Repetition Frequency) of a burst signal. Use the SENS:ACQ:HOFF :TIME command to fine tune the burst synchronization time. This time should be longer than the burst, but shorter than the burst spacing. This implies that the burst duration is less than 50% of the pulse repetition frequency (PRF).

How to measure bursts is described in detail in chapter 7 of the Operators Manual.

Query syntax: **:MEASure[:SCALar]:FREQuency:BURSt:PRF?**
[_[<expected value>[,<resolution>]][,<(@1|2|3|4|5|6|7)>]]

Where the optional:
- <expected value> is the expected PRF,
- <resolution> is the required resolution.
- <(@1|2|3|4|5|6|7)> is the channel to measure on:
  - (@1) means input A
  - (@2) means input B
  - (@3) means input C (HF-input option)
  - (@4) means input E (Rear panel arming input)
  - (@5) means input A prescaled by 2
  - (@6) means the internal 10 MHz reference
  - (@7) means input A with adjustable sensitivity

If you omit the channel, the instrument measures on input A.

The <expected value> and <resolution> are used to calculate the measuring time (ACQuisition:APERture).

Complies to standards: Philips Timer/Counter 1991.

Period

Period — query

A traditional period measurement is performed.

Query syntax: **:MEASure[:SCALar]:PERiod?**[␣[<expected value>[,<resolution>]][,<(@1|2|3|4|5|6|7)>]]

Where the optional:

<expected value> is the expected Period,
<resolution> is the required resolution,
<(@1|2|3|4|5|6|7)> is the channel to measure on:
(@1) means input A
(@2) means input B
(@3) means input C (HF-input option)
(@4) means input E (Rear panel arming input)
(@5) means input A prescaled by 2
(@6) means the internal 10 MHz reference
(@7) means input A with adjustable sensitivity

If you omit the channel, the instrument measures on input A.

The <expected value> and <resolution> are used to calculate the measuring time (ACQuisition:APERture).

Complies to standards: SCPI 1991.0, confirmed.

Time Interval — query

Traditional time interval measurements are performed. The trigger levels are set automatically and positive slope is used. The first channel in the channel list is the start channel and the second is the stop channel.

Query syntax: **:MEASure[:SCALar]:TINTerval?**[␣,(@1|2|4),(@1|2|4)]

Where the optional:

The first (@1|2|4) is the start channel and the second (@1|2|4) is the stop channel
(@1) means input A
(@2) means input B
(@4) means input E (Rear panel arming input)

If you omit the channel, input A is the start channel and input B is the stop channel.

Complies to standards: Philips Timer/Counter 1991.

Phase

Phase — query

A traditional PHASe measurement is performed.

Query syntax: **:MEASure[:SCALar]:PHASe?**[␣[<expected value>[,<resolution>]][,(@1|2),(@1|2)]]

Where the optional:

<expected value> and <resolution> are ignored by the counter

The first (@1|2) is the start channel and the second (@1|2) is the stop channel

(@1) means input A

(@2) means input B

If you omit the channel, the instrument measures between input A and input B.

Complies to standards: SCPI 1991-0, approved.

Pulse Width — query

A pulse width measurement is performed.

This is always a single measurement. If you need an average pulse width measurement, use the :SENSe command subsystem instead.

Query syntax Negative: **:MEASure[:SCALar]:NWIDth?**[_[<threshold>][,<(@1|2|4)>]]

A negative pulse width measurement is performed.

Query syntax Positive: **:MEASure[:SCALar]:PWIDth?**[_[<threshold>][,<(@1|2|4)>]]

A positive pulse width measurement is performed.

Where the optional:

<threshold> parameter sets the trigger levels in volts. If omitted, the auto trigger level is set to 50 percent of the signal.

<(@1|2|4)> is the channel to measure on:

(@1) means input A

(@2) means input B

(@4) means input E (Rear panel arming input)

If you omit the channel, the instrument measures on input A.

Complies to standards: SCPI 1991.0, confirmed.

Duty Cycle

Duty Cycle — query

Traditional duty cycle measurement is performed.

Query syntax, Positive: **:MEASure[:SCALar]:PDUTycycle|:DCYCle?** [_[<threshold>][,(@1|2)]]

Query syntax, Negative: **:MEASure[:SCALar]:NDUTycycle?**[_[<threshold>][,(@1|2)]]

Where the optional:

<threshold> parameter sets the trigger levels in volts. If omitted, the auto trigger level is set to 50 percent of the signal.

(@1|2) is the channel to measure on

(@1) means input A

(@2) means input B

If you omit the channel, the instrument measures between input A and input B.

Complies to standards: SCPI 1991.0, confirmed.

Transition Time

query

Traditional transition time measurement is performed.

The measurement is always a single measurement and the Auto-trigger is always on, setting the trigger levels to 10% and 90 % of the amplitude. If you need an average transition time measurement, or other trigger levels, use the :SENSe command subsystem and manually set trigger levels instead.

Query syntax Rise-time:

:MEASure[:SCALar]:RISE:TIME?[␣[<lower threshold>[,<upper threshold>[,<expected value>[,<resolution>]]]][,(@1)]]

Query syntax Fall-time:

:MEASure[:SCALar]:FALL:TIME?[␣[<lower threshold>[,<upper threshold>[,<expected value>[,<resolution>]]]][,(@1)]]

Where the optional:

<lower threshold> parameter sets the lower trigger level in percent of the signal (e.g. 10).
<upper threshold> parameter sets the upper trigger level in percent of the signal (e.g. 90). The PM 6680 ignores these parameters and uses the Auto-trigger to set the levels to 10% and 90 % of the amplitude.

<expected value> and <resolution> are ignored by the counter
<(@1)> is the channel to measure on, i.e., input A

If you omit the channel, the instrument measures on input A.

Complies to standards: SCPI 1991.0, confirmed.

Voltage — query

These commands make voltage measurements with the input DC coupled.

Query syntax, positive peak:

:MEASure[:SCALar]:MAXimum? [_(@1|@2)]

Where the optional:
(@1|@2) is the channel to measure on
(@1) means input A
(@2) means input B

A VOLT AC MAXimum measurement is performed.

Query syntax, negative peak:

:MEASure[:SCALar]:MINimum? [_(@1|@2)].

Where the optional:
(@1|@2) is the channel to measure on
(@1) means input A
(@2) means input B

A VOLT AC MINimum measurement is performed.

Query syntax, peak to peak:

:MEASure[:SCALar]:PTPeak?[_(@1|@2)]

Where the optional:
(@1|@2) is the channel to measure on
(@1) means input A
(@2) means input B

A VOLT AC Peak to Peak measurement is performed.

Complies to standards: SCPI 1991.0, confirmed.

Totalize X gated by Y — query

The counter totalizes the pulses on the primary channel. The totalizing starts when the gate signal on the secondary channel goes on and stops when the gate signal goes to off. The polarity of on/off is controlled via the INPut:SLOPe command of the gate signal.

Select if the counter should count positive or negative transitions with the INPut:SLOPe command of the measuring channel.

Query syntax: **:MEASure[:SCALar]:TOTalize:GATed?**

[_,(@1|@2|@4|@5|@6)[,(@1|@2|@4|@5|@6)]]

Where the optional:

First <(@1|@2|@4|@5|@6)> is the channel to measure on, the second one is the gate channel:

(@1) means input A
(@2) means input B
(@4) means input E (rear panel arming input)
(@5) means input A prescaled by 2
(@6) means the internal 10 MHz reference

If you omit the channels, the instrument measures on input A with input B as the gate channel.

Complies to standards: Philips Timer/Counter 1991.

Totalize X gated by Y, accumulated — query

The counter totalizes the pulses on the primary channel. The totalizing starts when the gate signal on the secondary channel goes on and stops when the gate signal goes to off. The polarity of on/off is controlled via the INPut:SLOPe command of the gate channel.The result is the sum of counts in all the gate openings that occur during a preset time <time for gate open>.

Select if the counter should count positive or negative transitions with the INPut:SLOPe command of the measuring channel.

Query syntax: **:MEASure[:SCALar]:TOTalize:ACCumulated?**[_<time for gate open>][,(@1|2|4|5|6) [,(@1|2|4|5|6)]]

Where the optional:

<time for gate open> is the time you want the totalizing to proceed. The range is 0.8E–6, 1.6E–6, 3.2E–6, 6.4E–6, 12.8E–6, and 50E–6 to 400 s.

The first <(@1|2|4|5|6)> is the channel to measure on.
The second <(@1|2|4|5|6)> is the gate channel.
(@1) means input A
(@2) means input B
(@4) means input E (rear panel arming input)
(@5) means input A prescaled by 2
(@6) means the internal 10 MHz reference

NOTE: If you omit the channels, the instrument measures on input A with input B as the gate channel.

✷RST condition: Time for gate open = 10 ms (ACQuisition:APERture)

Complies to standards: Philips Timer/Counter 1991.

Totalize X start/stop by Y — query

The counter totalizes the pulses on the primary channel. The totalizing starts when the gate signal on the secondary channel goes on and stops the next time the gate signal goes on. The polarity of ON is controlled via the INPut:SLOPe command of the Start /stop channel.

Select if the counter should count positive or negative transitions with the INPut:SLOPe command of the measuring channel.

Query syntax: **:MEASure[:SCALar]:TOTalize:SSTop?**[_,(@1|2|4|5|6)[,(@1|2|4|5|6)]]

Where the optional:

The first <(@1|2|4|5|6)> is the channel to measure on and the second one is the start/stop channel:

(@1) means input A
(@2) means input B
(@4) means input E (rear panel arming input)
(@5) means input A prescaled by 2.
(@6) means the internal 10 MHz reference.

If you omit the channels, the instrument measures on input A with input B is the start/stop channel.

Complies to standards: Philips Timer/Counter 1991.

Totalize X-Y During a Preset Time

Totalize X-Y During a Preset Time — command

This is a count/totalize function during a pre-defined time. The start/stop signal is generated by the counter and set by <time for gate open>.

The counter counts up for each event on the X-channel and down for each event on the Y-channel. The result is the difference between the two channels.

If you only want to Totalize on X, you must disable Y by: setting the Y-channel to (@0) or, setting both X and Y to the same channel, or disconnecting the signal from Y.

Totalize –Y MANUAL, negative totalizing, is possible if you set X to (@0) or physically disconnect the signal on the X input.

Select if the counter should count positive or negative transitions with the INPut:SLOPe command of the channels.

Command syntax: **:MEASure[:SCALar]:TOTalize:TIMed?**[_[<time for gate open> [,(@0|1|2|4)][,(@0|1|2|4)]]]

Where the optional:

<time for gate open> is the time you want the totalizing to proceed. The range is 0.8E–6, 1.6E–6, 3.2E–6, 6.4E–6, 12.8E–6 and 50E–6 to 400 s.

The first <(@0|1|2|4)> is the channel that counts up and the second one is the channel that counts down:
(@0) means no signal to this channel
(@1) means input A
(@2) means input B
(@4) means input E (rear panel arming input)

If you omit the channels, the instrument counts up on input A and down on input B.

****RST condition:*** Time for gate open = 10 ms (ACQuisition APERture)

Complies to standards: Philips Timer/Counter 1991.

Totalize X–Y Manually — command

This is a count/totalize function controlled from the GPIB interface using the command SENS:TOT:GATE␣ON|OFF.

The counter counts up for each event on the X-channel and down for each event on the Y-channel. The result is the difference between the two channels.

If you only want to Totalize on X, you must disable Y by setting the Y-channel to (@0), or you must disconnect the signal from Y.

Totalize –Y MANUAL, negative totalizing, is possible if you set X to (@0) or physically disconnect the signal on the X input.

Select counting on positive or negative transitions with the INPut:SLOPe command for the used channels.

The ability to accept negative parameters for the FETCh command was added to the PM 6680 firmware 1.02 and PM 9626 (GPIB) firmware 1.11. If you need this function but the firmware in your counter is older, your Fluke/Philips service center can upgrade it for you.

Command syntax:

:CONFigure[:SCALar]:TOTalize:CONTinuous
[␣,(@0|1|2|4)[,(@0|1|2|4)]]

Where the optional:
<(@0|1|2|4)> is the channel to measure on:
(@0) means no signal to this channel
(@1) means input A
(@2) means input B
(@4) means input E (rear panel arming input)

NOTE: This measurement cannot be done as a :MEASure, it must be done as a :CONFigure followed by INIT:CONT␣ON, gate control with SENS:TOT:GATE␣ON|OFF and completed with a :FETCh:ARR?.

Complies to standards: Philips Timer/Counter 1991.

Totalize X–Y Manually

Setting up totalizing

Normal program sequence for totalizing on A	
CONF:TOT:CONT (@1),(@0)	Set up the counter for totalize on A and ignore subtraction of pulses on B
INIT:CONT ON	Initiate the counter continuously
TOT:GATE ON	Start totalizing
FETC:ARR? –1	Read intermediate results without stopping the totalizing
TOT:GATE OFF	Stops totalizing
FETC:ARR? –1	Fetch the final result from the totalizing

The FETCh:ARR? command can take both positive and negative data. Positive data, for instance 10, outputs the first 10 measurements in the PM 6680 output buffer. Negative data, for instance–10 outputs the last ten results.

Intermediate results

When totalizing you often want to read intermediate result without stopping the totalizing process. FETc:ARR? –1 outputs such a result.

How to use the Measure function

The Measure function group has a different level of compatibility and flexibility than other commands. The parameters used with commands from the Measure group describe the signal you are going to measure. This means that the Measure functions give compatibility between instruments, since you don't need to know anything about the instrument you are using.

MEASure?

This is the most simple command to use, but it does not offer much flexibility. The MEASure? command lets the instrument configure itself for an optimal measurement, starts the data acquisition, and returns the result.

● *Example:* SEND→ **MEASure:FREQ?**

This will execute a frequency measurement and the result will be sent to the controller. The instrument will select a setting for this purpose by itself, and will carry out the required measurement as "well" as possible; moreover, it will automatically start the measurement and send the result to the controller.

You may add parameters to give more details about the signal you are going to measure, for example:

SEND→ **MEASure:FREQ?␣20␣MHZ,1**

Where: 20␣MHZ is the expected value, which can of cause also be sent as 20E6, and 1 is the required resolution. (1 Hz)

Also the channel numbers can be specified, for example:

SEND→ **MEASure:FREQ?␣(@2)**

SEND→ **MEASure:FREQ?␣20E6,1,(@2)**

CONFigure; READ?

The CONFigure command causes the instrument to choose an optimal setting for the specified measurement. CON-

Figure may cause any device setting to change. READ? starts the acquisition and returns the result.

This sequence operates in the same way as the MEASure command, but now it is possible to insert commands between CONFigure and READ? to fine tune the setting of a particular function, e.g., you can set an input attenuator at a required value.

● *Example:* SEND→ **CONFigure:FREQ␣2E6,1**

1 is the required resolution

2E6 is the expected value

SEND→ **INPut:IMPedance␣1MOHM**

Sets input impedance to 1 MOhm

SEND→ **READ?**

Starts measurement and returns the result.

CONFigure; INITiate;FETCh?

The READ? command can be divided into the INITiate command, which starts the measurement, and the FETCh? command, which requests the instrument to return the measuring results to the controller.

● *Example:* SEND→ **CONFigure:FREQ␣20E6,1**

1 is the required resolution

20E6 is the expected signal value

SEND→ **INPut:IMPedance␣1E6**

Sets input impedance to 1 MOhm

SEND→ **INITiate**

Starts measurement

SEND→ **FETCh?**

Versatility of measurement commands	
MEASure?	Simple to use, few additional possibilities.
CONFigure READ?	Somewhat more difficult, but some extra possibilities.
CONFigure INITiate FETCh?	Most difficult to use, but many extra features.

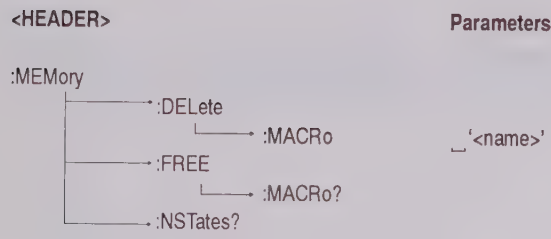
This page is intentionally left blank.

Chapter 12

Memory Subsystem

Memory Subsystem Command Tree

Memory Subsystem Command Tree



✱SAV
✱RCL
✱LRN?
✱DMC
✱EMC
✱GMC?
✱LMC?
✱PMC

Delete one Macro command

This command removes an individual MACRo[1].

Command syntax: **:MEMory:DELete:MACRo**␣'<macro name>'

Where:

'<macro name>' is the name of the macro you want to delete.

NOTE: <macro name> is String data that must be surrounded by quotation marks.

See also: ✱PMC, if you want to delete all macros.

Complies to standards: SCPI 1991.0, confirmed.

1 The proposed IEEE488.2 command ✱RMC (Remove Macro command) also works on PM 6680. It preforms exactly the same action as :MEMory:DELete:MACRo. Note however that this command is not yet (1991) a certified IEEE488.2 command.

Memory Free for Macros

Memory Free for Macros — query

This command gives information of the free memory available for MACRos in the counter. If no macros are specified, 1160 bytes are available.

Query syntax: **:MEMory:FREE:MACRo?**

- *Returned format:* <bytes available>, <bytes used>↵

Complies to standards: SCPI 1991.0, confirmed.

Memory states

query

The Number of STates query (only) requests the number of ✷SAV/✷RCL instrument setting memory states available in the counter. The counter responds with a value which is one greater than the maximum that can be sent as a parameter to the ✷SAV and ✷RCL commands. (States are numbered from 0 to max–1.)

Query syntax: **:MEMory:NSTates?**

● *Returned format:* <the number of states available>↵

Complies to standards: SCPI 1991.0, confirmed.

Save

Save **command**

Saves the current settings of the instrument in an internal nonvolatile memory. Twenty memory locations are available. Switching the power off and on does not change the settings stored in the registers.

Note that memory positions 10 to 19 can be protected from the front panel auxiliary menu. If this has been done, use the :SYSTem:UNPRotect command, then you can alter these memory positions.

Command Syntax: ✷**SAV**␣<decimal data>

Where:

<decimal data> = a number between 0 and 19.

● *Example:* ✷SAV␣10↵

Complies to standards: IEEE 488.2 1987.

Recall

command

Recalls one of the up to 20 previously stored complete instrument settings from the internal nonvolatile memory of the instrument.

- *Command Syntax:* ✱**RCL**␣<decimal data>

  Where:

  <decimal data> = a number between 0 and 19.

- *Example:* ✱RCL␣10↵

Complies to standards: IEEE 488.2 1987.

Learn Device Setup — query

Learn Device Setup Query. Causes a response message which can be sent to the instrument to return it to the state it was in when the ✷LRN? query was made.

Query Syntax: **✷LRN?**

- *Returned Format:* :SYST:SET␣<block data>

  Where <block data> consists of:
  #292<92 data bytes>

- *Example* SEND ✷LRN?

Complies to standards: IEEE 488.2 1987.

Define Macro — command

Allows you to assign a sequence of one or more program message units to a macro label. The sequence is executed when the macro label is received as a command or query. Twentyfive macros can be defined at the same time and each macro can contain an average of 40 characters.

If macro has the same name as a command, then it masks out the real command with the same name when macros are enabled. If macros are disabled, the original command will be executed.

If you define macros when macro execution is disabled, the counter executes the ✱DMC command fast, but if macros are enabled, the execution time for this command is longer.

Command Syntax: ✱DMC␣<macro label> , <program messages>

Where:

<macro label> = one to 12 character macro label. (String data must be surrounded by " " or ' ' as in the example below.)

<program messages> = the commands to be executed when the macro label is received, both block data and string data formats can be used.

● *Example 1:* SEND→ `✱DMC␣'MyInpSetting',#241:INP1:IMP␣50;COUP␣DC;:INP2:IMP␣50;COUP␣DC`

This example defines a macro MyInpSetting which sets both A and B inputs to 50 Ω and DC.

● *Example 2:* SEND→ `✱DMC␣'MyInpSetting',#241:INP1:IMP␣$1;COUP␣$2;:INP2:IMP␣$1;COUP␣$2`

This example defines a macro MyInpSetting which takes two arguments, i.e., the Impedance($1) and the Coupling.($2).

SEND→ `MyInpSetting␣1MOHM,AC`

sets the input impedance to 1 MΩ and the coupling to AC on both inputs.

Complies to standards: IEEE 488.2 1987.

Enable Macros

Enable Macros — command/query

This command enables and disables expansion and execution of macros. If macros are disabled, the instrument will not recognize a macro although it is defined in the instrument. (The Enable Macro command takes a long time to execute.)

Command Syntax: ✷EMC␣<dec. data>

Where:

<dec. data> = is 0 or 1. A value which rounds to 0 turns off macro execution. Any other value turns macro execution on.

● *Example:* SEND→ ✷EMC␣1 ↵

Enable macro expansion and execution.

Query Syntax: ✷EMC?

● *Returned Format:* 0|1 ↵

1 indicates that macro expansion is enabled.

0 indicates that macro expansion is disabled.

Complies to standards: IEEE 488.2 1987.

Get Macro

query

This command makes the counter respond with the current definition of the given macro label.

Query Syntax: **✷GMC?**␣< macro label>

Where:

<macro label> = the label of the macro for which you want to see the definition. (String data must be surrounded by " " or ' ' as in the example below.)

- *Returned Format:* <block data>↵
- *Example:* SEND→ `✷GMC?␣'MyInpSetting'↵`

may give the following response:

READ→ `#241:INP1:IMP␣$1;COUP␣$2;`
`:INP2:IMP␣$1;COUP␣$2↵`

Complies to standards: IEEE 488.2 1987.

Learn Macro — query

Makes the instrument output a list of string data elements, containing all macro labels defined in the instrument.

Query Syntax: **✱LMC?**

- *Returned Format:* <string> { ,<string> }↵

  Where:

  <string> = a Macro label. (String data will be surrounded by " " as in the example below.)

- *Example:*

SEND→ **✱LMC?** ↵

may give the following response:

READ→ **"MYINPSETTING","LIMITMON"**↵

Complies to standards: IEEE 488.2 1987.

Purge Macros

command

Removes all macro definitions.

Command Syntax: **✱PMC**

- *Example:* ✱PMC ↵

See also: :MEMory:DELete:MACRo␣'<macro name>' if you want to remove a single macro.

Complies to standards: IEEE 488.2 1987.

This page is intentionally left blank.

How to use Macros

A macro is a single command, which represents one or several other commands, depending on your definition. You can define 25 macros of 40 characters in the PM 6680. One macro can address other macros, but you cannot call a macro from within itself (recursion). You can use variable parameters that modify the macro.

Use macros to:

Provide a shorthand for complex commands.

Cut down on bus traffic.

- *Macro names*

You can use both commands and queries as macro labels. The label cannot be the same as common commands or queries. If a macro label is the same as a PM 6680 command, the PM 6680 will execute the macro when macros are enabled (*EMC 1) and it will execute the PM 6680 command when macros are disabled (*EMC 0).

- *Data types within macros*

The commands to be performed by the macro can be sent both as block and string data.

String data is the easiest to use since you don't have to count the number of characters in the macro. However, there are some things you must keep in mind:

Both double quote (") and single quote (') can be used to identify the string data. If you use a controller language that uses double quotation marks to define strings within the language (like BASIC) we recommend that you use block data instead, and use single quotes as string identifiers within the macro.

CAUTION: When using string data for the commands in a macro, remember to use a different type of string data identifiers for strings within the macro. If the macro should for instance set the input slope to positive and select the period function, you must type

```
":Inp:slope pos;Func 'PER 1'"
```

or

```
':Inp:slope pos;Func "PER 1"'
```

How to use Macros

Define macro command

✱DMC assigns a sequence of commands to a macro label. Later when you use the macro label as a command, the Timer/Counter will execute the sequence of commands.

Use the following syntax:

✱DMC <macro-label>, <commands>

- *Simple macros*

Example:

SEND→ `✱DMC␣'MYINPSETTING',':INP1:IMP␣50 ;COUP␣DC;:INP2:IMP␣50;COUP␣DC'8Mƒ`

This defines a macro named MYINPSETTING which sets the input impedance to 50 Ω and the coupling to DC on both A and B inputs.

- *Macros with arguments*

You can pass arguments (variable parameters) with the macro. Insert a dollar sign ($) followed by a single digit in the range 1 to 9 where you want to insert the parameter. See the example below.

When a macro with defined arguments is used, the first argument sent will replace any occurrence of $1 in the definition; the second argument will replace $2 etc.

Example:

SEND→ `✱DMC␣'MYINPSETTING',':INP1:IMP␣$1;COUP␣$2;:INP2:IMP␣$1;COUP␣$2'128Mƒ`

SEND→ `MYINPSETTING␣1MOHM,AC`

will set the input impedance to 1MΩ and the coupling to AC for both A and B inputs.

Deleting macros

Use the **✱PMC (purge macro)** command to delete all macros defined with the ✱DMC command. This removes all macro labels and sequences from the memory. To delete only one macro in the memory, use the :MEMory:DELete:MACRo command.

NOTE: You cannot overwrite a macro, you must delete it before you can use the same name for a new macro.

Enabling and disabling macros

***EMC Enable macro command**

When you want to execute a PM 6680 command or query with the same name as a defined macro, you need to disable macro execution. Disabling macros does not delete stored macros; it just hides them from execution.

Disabling: *EMC 0 disables all macros.
Enabling: *EMC 1

***EMC? Enable Macro Query**

Use this query to determine if macros are enabled.

Response:
1 macros are enabled
0 macros are disabled

How to execute a macro?

Macros are disabled after *RST, so to be sure, start by enabling macros with *EMC 1. Now macros can be executed by using the macro labels as commands.

● *Example:* SEND→ `*DMC 'LIMITMON',':CALC:STAT ON;:CALC:LIM:STAT ON;:CALC:LIM:LOW:DATA $1;STAT ON;:CALC:LIM:UPP:DATA $2;STAT ON'<NL>`

SEND→ `*EMC 1`

Now sending the command:

SEND→ `LIMITMON 1E6,1.1E6`

will switch on the limit monitoring, assign upper and lower limits and switch on the limit monitor.

Retrieve a macro

- ✱*GMC? Get macro contents query*

  This query gives a response containing the definition of the macro you specified when sending the query.

  Example using the above defined macro:

  SEND→ `✱GMC? 'LIMITMON'`

  READ← `#292:CALC:STAT_ON ;:CALC:LIM:STAT_ON ;:CALC:LIM:LOW:DATA_$1;STAT_ON ;:CALC:LIM:UPP:DATA_$2;STAT_ON'`↵5D

- ✱*LMC? Learn macro query*

  This query gives a response containing the labels of all the macros stored in the Timer/Counter.

  Example:

  SEND→ `✱LMC?`

  READ← `"MYINPSETTING","LIMITMON"`

  Now there are two macros in memory, and they have the following labels: "MYINPSETTING" and "LIMITMON".

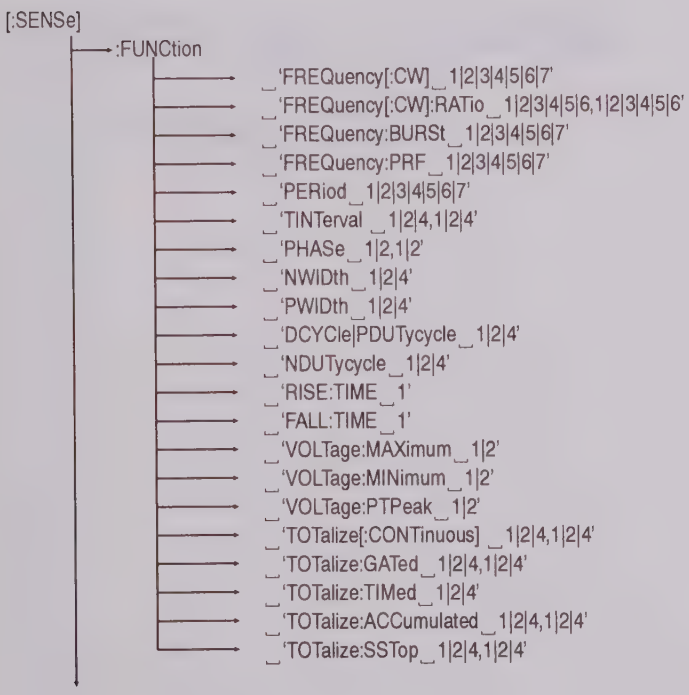
Chapter 13

Sense Command Subsystems

Sense Subsystem Command Tree

Sense Subsystem Command Tree

<HEADER> Parameters



Continued on the next page

Sense Subsystem Command Tree

<HEADER	Parameters
[:SENSe] (Continued from previous page.)	
:ACQuisition	
:APERture	<meas time>|MIN|MAX
:HOFF	
[:STATe]	<boolean>
:MODE	TIME|EVENT
:TIME	<hold off time value>|MIN|MAX
:ECOunt	<hold off event value>|MIN|MAX
:RESolution	HIGH|LOW
:AVERage	
:STATe	<boolean>
:COUNt	<numeric value>|MIN|MAX
:MODE	TIME|COUNts
:FREQuency	
:RANGe	
:LOWer	<numeric value>|MIN|MAX
:ROSCillator	
:SOURce	INTernal|EXTernal
:TOTalize	
:GATE	
[:STATe]	<boolean>
:VOLTage	
:GATed	
:STATe	<boolean>
:INTernal	
:FORMat	REAL|PACKed

Input channels 1|2|3|4|5|6|7 is the channel to measure on:

- 1 means input A
- 2 means input B
- 3 means input C (HF-input option)
- 4 means input E (Rear panel arming input)
- 5 means input A prescaled by 2.
- 6 means the internal 10 MHz reference.
- 7 means input A with adjustable sensitivity.

Select Measuring Function

Select Measuring Function — command/query

Selects which measuring function is to be performed and on which two channels the instrument should measure.

Command syntax: **[:SENSe]:FUNCtion** ␣'<measuring function> [␣<primary channel>,<Secondary channel>]'

Where: <measuring function> is the function you want to select, according to the SENSe subsystem command tree on page 13-2.
<primary channel> is the channel used in all single-channel measurements and the main channel in dual-channel measurements.
<secondary channel> is the 'other' channel in dual-channel measurements. Only the primary channel needs to be programmed for all single channel measurements.

NOTE: All parameters and channels together form one <string> that must be placed within quotation marks.

● *Example* Select a time interval measurement between input B (channel 2) and input E (channel 4):

Send → `:FUNC␣'TINT␣2,4'`

Query syntax: [:SENSe]:FUNCtion?

● *Returned format:* "<measuring function>␣<primary channel>[,<secondary channel>]"↵

****RST condition:*** FREQuency␣1

Complies to standards: SCPI 1991.0, confirmed.

Set the Measuring-Time — command/query

Sets the gate time for one measurement.

Parameter range: 800 ns, 1.6, 3.2, 6.4, 12.8 µs and 50 µs to 400 s for all measuring functions. Measuring times of 800 ns to 12.8 µs work in FREQ:CW, FREQ:BURST, FREQ:PRF and PERiod. If one of these short times is selected when the counter makes other measurements it will use 50 µs.

NOTE: if you want to switch between Average and Single measurements, use the :AVERage:STATe␣ON|OFF in the Sense Subsystem.

When Single is selected and an array measurement is done, the Measuring time, set by :Acquisition:APERture, sets the time between the measurements in the array. This means that if you want a very high speed yo must set AVER:STATE to off and ACQ:APER to a short measuring time.

Command syntax: **[:SENSe]:ACQuisition:APERture**␣<Decimal value > |min|max

Where:

<decimal value> is 0.8E–6, 1.6E–6, 3.2E–6, 6.4E–6, or 12.8E–6 and 50E–6 to 400.

Query syntax: [:SENSe]:ACQuisition:APERture?

● *Returned format:* < Decimal value >↵

✷RST condition: 10 ms

Complies to Standards: Philips Timer/Counter 1991

Hold Off on/off

Hold Off on/off — command/query

Switches the Hold Off function on/off.

Command syntax: **[:SENSe]:ACQuisition:HOFF[:STATe]** <boolean>

Where:
<Boolean> = 1/ON|0/OFF

Query syntax: [:SENSe]:ACQuisition:HOFF[:STATe]?

- *Returned format:* 1 | 0↵
- **RST condition:* OFF

Complies to standards: Philips Timer/Counter 1991.

Hold Off Mode — command/query

Selects if triggering is going to be disabled for a preset time or for a preset number of events.

When set to event, the counter counts negative edges on the B input (channel 2).

This function is coupled to the :ARM:START:DEL, :ARM:START:ECO, ARM:STOP:DEL and :ARM:STOP:ECO. The different delays must all be of the same type, (Time or Event). This means that setting one of them to Event delay causes the others to be set to Event delays.

Command syntax: **[:SENSe]:ACQuisition:HOFF:MODE** TIME|EVENt

Query syntax: [:SENSe]:ACQuisition:HOFF:MODE?

● *Returned format:* TIME| EVENt↵

✷RST condition: TIME

Complies to standards: Philips Timer/Counter 1991.

Hold Off Time

Hold Off Time — command/query

Sets the Hold Off time value.

Parameter range: 200 ns to 1.677 s

Command syntax: **[:SENSe]:ACQuisition:HOFF:TIME** <Decimal value>|min|max

Where:
<decimal data> = a number between 200E–9 and 1.6777215.

Query syntax: [:SENSe]:ACQuisition:HOFF:TIME?

● *Returned format:* <Decimal value>↵

RST condition: 10 µs

Complies to standards: Philips Timer/Counter 1991.

Hold Off Events — command/query

Sets the Hold Off event value. The counter counts negative events on the B input (channel 2).

Parameter range:
2 to (2^{24} –1) counts.

Command syntax: **[:SENSe]:ACQuisition:HOFF:ECOunt** <Decimal value>|min|max

Where:
<decimal value> is a number in the range 2 to 16 777 215.

Query syntax: [:SENSe]:ACQuisition:HOFF:EVENt?

● *Returned format:* <Decimal value>↵

✷RST condition: 100

Complies to standards: Philips Timer/Counter 1991.

Resolution

Resolution — command/query

Selects basic measurement mode for all time related measurements. When HIGH the resolution is the full 0.5 ns, and with LOW the resolution is limited to a 100 ns clock. You can use this to increase the bus speed. Saves about 0.6 to 0.9 ms if the counter does real time calculations, otherwise only 0.05 ms.

Command syntax: **[:SENSe]:ACQuisition:RESolution**␣ HIGH|LOW

Query syntax: [:SENSe]:ACQuisition:RESolution?

● *Returned format:* HIGH|LOW↵

RST condition: HIGH

Complies to standards: Philips Timer/Counter 1991.

Average or Single? command/query

Switch on/off the average function.

ON means multiple period measurements for period related measurements and time interval average for time interval measurements.

OFF means that the counter measures on a single cycle. This is the same as when pressing the SINGLE key on the front panel.

When Single is selected and an array measurement is done, the Measuring time, set by :Acquisition:APERture, sets the time between the measurements in the array. This means that if you want a very high speed yo must set AVER:STATE to off and ACQ:APER to a short measuring time.

Command syntax: **[:SENSe]:AVERage:STATe** ␣<Boolean>

Where:
<boolean> = 1|ON/0|OFF

Query syntax: [:SENSe]:AVERage:STATe?

● *Returned format:* <Boolean>↵

****RST condition:*** ON

Complies to Standards: Philips Timer/Counter 1991.

Average Samples

Average Samples — command/query

Sets the number of samples to use when doing time interval averaging measurements. Applies to the functions: PWIDTH, TIME, RISE and FALL TIME.

Command syntax: **[:SENSe]:AVERage:COUNt** <decimal data>|min| max

Where:
<decimal data> is a number between 1and 65535.

Query syntax: [:SENSe]:AVERage:COUNt?

- *Returned format:* <decimal data>↵

****RST condition:*** 100

Complies to standards: Philips Timer/Counter 1991.

Average Mode — command/query

Selects whether the gate time (:SENS:ACQ:APER) or a preset number of events (:SENS:AVER:COUN) is going to limit the umber of samples when doing time interval averaging measurements (PWIDTH, TIME, RISE and FALL TIME).

Command syntax: **[:SENSe]:AVERage:MODE** _ TIME|COUNts

Query syntax: [:SENSe]:AVERage:MODE?

● *Returned format:* TIME|COUN↵

****RST condition:*** TIME

Complies to standards: Philips Timer/Counter 1991.

High Speed Voltage Measurements — command/query

Use this command to speed up voltage measurements and auto trigger functions on signals above 10 kHz.

The VOLT and auto trigger functions can determine the level quicker when the frequency of the input signal is above 10 kHz.

Time to determine trigger levels		
Measuring function	**100 Hz frequency limit (normal)**	**10 kHz frequency limit**
Freq A	48 ms	26 ms
Time A-B	82 ms	38 ms

Command syntax: **[:SENSe]:FREQuency:RANGe:LOWer** <numeric value>|MIN|MAX

Where:

<numeric value> is 100 for a lower frequency limit of 100 Hz
and 10000 for a lower frequency limit of 10 kHz.
MIN gives 100 Hz frequency limit.
MAX gives 10 kHz frequency limit.

Query syntax: [:SENSe]:FREQuency:RANGe:LOWer?

● *Returned format:* <numeric value>↵

✷RST condition: 100

Complies to standards: SCPI 1991.0, confirmed.

Select Oscillator Source — command/query

Select the signal from the external reference input as clock instead of the internal oscillator.

Command syntax: **[:SENSe]:ROSCillator:SOURce**␣<INT|EXT>

Query syntax: [:SENSe]:ROSCillator:SOURce?

● *Returned format:* <INT|EXT>↵

✷***RST condition:*** INT

Complies to standards: SCPI 1991.0, confirmed.

Gate on/off

Gate on/off — command/query

Open/closes the gate for the TOTalize[:CONTinuous].

NOTE: Before opening the gate with this command, the counter must be in the 'continuously initiated' state (:INIT:CONT ON), or else the totalizing will not start.

Command syntax: **[:SENSe]:TOTalize:GATE[:STATe]** _<Boolean>

Where:

<Boolean> = 1/ON|0/OFF

Query syntax: [:SENSe]:TOTalize:GATE[:STATe]?

● *Returned format:* <Boolean>↵

✷RST condition: OFF

Complies to standards: Philips Timer/Counter 1991.

Gated Voltage Measurement — command/query

Selects the gated mode for the VOLTage:MAX|MIN|PTPeak measuring functions and for the AUTO Trigger function.

The gated mode is useful for removing over- and undershoot. The gate signal is controlled by the ARM:STOP :SLOPe and ARM:STOP:SOURce commands. If channel 2 (B) is the source for the gating signal, then all other characteristics of that channel can be used. When Gated Voltage is selected, the Stop Arming function is disabled from its normal stop arming usage. When gated voltage mode is selected, high enables measurement and low disables measurements. Use the slope if you want it the other way around.

Command syntax: **[:SENSe]:VOLTage:GATed:STATe** <Boolean>
Where:
<Boolean> = 1/ON|0/OFF

Query syntax: [:SENSe]:VOLTage:GATed:STATe?

● *Returned format:* 1|0 ↵

✷RST condition: OFF

Complies to standards: Philips Timer/Counter 1991.

Internal Format **command/query**

This command selects the internal data format of the measurement result from the SENSe block. This command is used to increase the measurement speed. PACKed means that the raw measurement data is stored internally and the result is not calculated in real time between measurements. The results are calculated later when they are sent to the controller. Since the result is not calculated, other blocks cannot use this data. That means that you cannot have the DISPlay, OUTPut and CALCulate blocks switched on then using PACKed format. The following measuring functions cannot be used with PACKed format: Time interval average measurements (TINT, RISE, FALL, PWID) PHASe and DUTY.

Command syntax: **:INTernal:FORMat** _REAL|PACKed

Query syntax: :INTernal:FORMat?

✷RST condition: REAL

Complies to standards: Philips Timer/Counter 1991.

How to Select Input

Depending on application, you can select different input channels and input characteristics.

● *Switchbox*

In automatic test systems, it is important that you can switch between different signal sources without swapping BNC cables. You may select from the three different basic frequency inputs (A, B, and E), on which the counter can measure directly without the need for external switching devices.

● *Prescaling*

For all measuring modes except FREQUENCY, the maximum input signal repetition rate for inputs A and B is 160 MHz.

To extend the range in the FREQUENCY mode, PM 6680 can divide (scale) the input frequency on input A by 2. When using channel 1, the counter automatically selects the most appropriate scaling factor depending on the functions selected. When measuring frequency on channel 1, the timer/counter divides the frequency by 2 giving 225 MHz max frequency, unless you use the Hold Off function. In all other measuring modes, it does not divide the signal and the max repetition rate is 160 MHz. Use channel 2 (via input B) if, for any reason, you don't want to divide the signal when measuring frequency.

Characteristics for different input channels

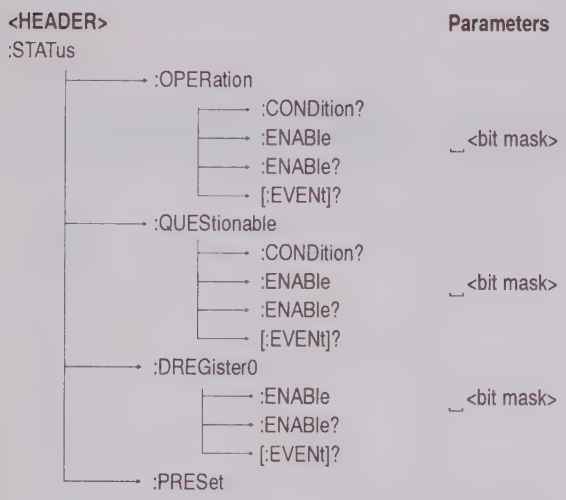
Channel	Input	Max frequency	Divider factor: d	Comments
1	A	225 MHz	2	Divides by 2 only in frequency mode when Hold Off is switched off, otherwise divides by 1.
2	B	160 MHz	1	Always used on input B.
3	C	1.3 GHz	256	Option PM 9621.
3	C	2.7 GHz	16	Option PM 9624.
3	C	4.5 GHz	32	Option PM 9625.
4	E	50 MHz	1	Rear panel input. TTL-levels.

Characteristics for different input channels				
Channel	**Input**	**Max frequency**	**Divider factor: d**	**Comments**
5	A	160 MHz	2	Can be used in all modes, except frequency and in frequency mode when using Hold Off.
6	—	10 MHz	—	Measures on the internal 10 MHz reference.
7	A	120 MHz	1	Use for frequency measurements to get variable sensitivity. This selection disables the B-input. The hysteresis band of input-A is now between the Input-A and Input-B trigger levels.

Chapter 14

Status Subsystem

Status Subsystem Command Tree



*CLS
*ESE
*ESR?
*PSC
*SRE
*STB?

Operation Status, Condition — query

Reads out the contents of the operation status condition register. This register reflects the state of the measurement process. See figure below. Note that bits 0 to 3, 7, and 9 to 15 are not used.

Query Syntax: **:STATus:OPERation:CONDition?**

- *Returned Format:* <decimal data> ↵

  Where:

  <decimal data> = an integer according to table below:

Bit No.	Weight	Condition
8	256	Not Measurement
6	64	Waiting for bus arming
5	32	Waiting for triggering and/ or external arming
4	16	Measurement

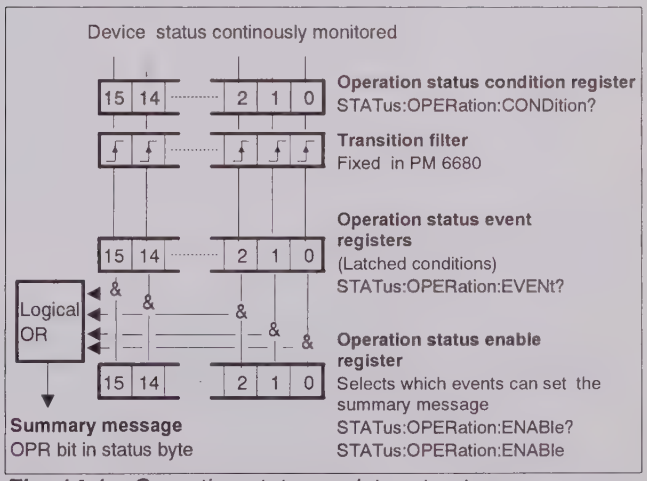


Fig. 14-1 *Operation status register structure.*

Complies to standards: SCPI 1991.0, confirmed

Operation Status, Enable — command/query

Sets the enable bits of the operation status enable register. This enable register contains a mask value for the bits to be enabled in the operation status event register. A 1 in the enable register enables the corresponding bit in the status register. See figure on page 14-3.

An enabled bit will set bit #7, OPR (Operation Status Bit), in the Status Byte Register if the enabled event occurs. See also status reporting on page 14-18.

This register will be cleared at power on if power on clearing is enabled via ✱PSC.

Command Syntax: **:STATus:OPERation:ENABle** _ <dec. data>

Where:

<dec.data> = an integer number according to the table below:

Bit No.	Weight	Condition
8	256	Measurement or block of measurements stopped
6	64	Waiting for bus arming
5	32	Waiting for triggering and/or external arming
4	16	Measurement or block of measurements started

● *Example:* SEND→ `:STAT:OPER: _288`↵

In this example waiting for triggering, bit 5, and Measurement stopped, bit 8, will set the OPR-bit of the Status Byte. (This method is faster than using ✱OPC if you want to know when the measurement is ready.)

Query Syntax: :STATus:OPERation:ENABle?

● *Returned Format:* <dec. data>↵

Complies to standards: SCPI 1991.0, confirmed.

Operation Status, Event query

Reads out the contents of the operation event status register. Reading the Operation Event Register clears the register. See figure on page 14-3.

Query Syntax: **:STATus:OPERation[:EVENt]?**

- *Returned Format:* <dec. data>↵

  Where:

  <dec. data> = an integer number according to the table on page 14-4.

Complies to standards: SCPI 1991.0, confirmed.

Questionable Data/signal, Condition — query

Reads out the contents of the status questionable condition register.

Query Syntax: **:STATus:QUEStionable:CONDition?**

- *Returned Format:* <dec. data> ↵

  Where:

  <dec. data> = an integer number according to the table below.

Bit No.	Weight	Condition
14	16384	Unexpected parameter
10	1024	Time out
9	512	Overflow

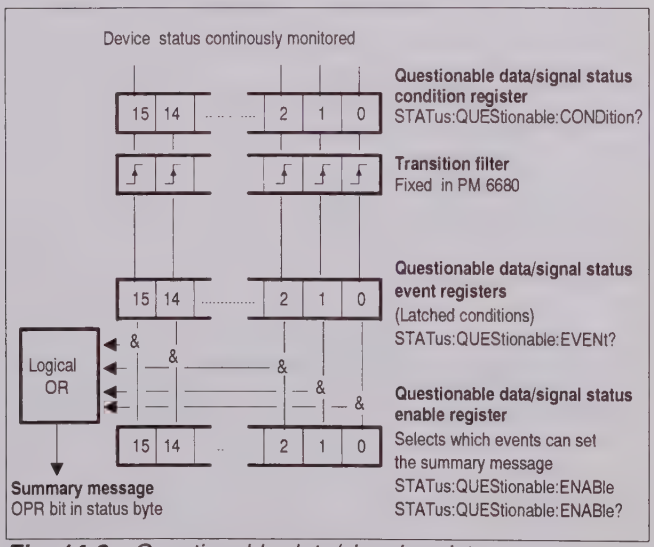
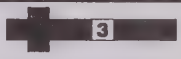


Fig. 14-2 Questionable data/signal register structure.

Complies to standards: SCPI 1991.0, confirmed.



Questionable Data/signal, Enable — command/query

Sets the enable bits of the status questionable enable register. This enable register contains a mask value for the bits to be enabled in the status questionable event register. A 1 in the enable register enables the corresponding bit in the status register. See figure on page 14-6.

An enabled bit will set bit #3, QUE (Questionable Status Bit), in the Status Byte Register if the enabled event occurs. See also status reporting on page 14-18.

This register will be cleared at power on if power on clearing is enabled via ✱PSC.

Command Syntax: **:STATus:QUEStionable:ENABle**␣<dec. data>

Where:

<dec.data> = an integer number according to the table on page 14-6.

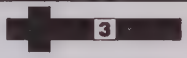
● *Example:* Send → **:STAT:QUES:ENAB␣16896**↵

In this example both 'unexpected parameter' bit 14, and 'overflow' bit 8, will set the QUE-bit of the Status Byte when a questionable status occurs.

Query Syntax: :STATus:QUEStionable:ENABle?

● *Returned Format:* <dec. data>↵

Complies to standards: SCPI 1991.0, confirmed.



Questionable Data/signal, Event — query

Reads out the contents of the status questionable event register. Reading the Status Questionable Event Register clears the register. See figure on page 14-6.

Query Syntax: **:STATus:QUEStionable[:EVENt]?**

- *Returned Format:* <dec. data> ↵

  Where:

  <dec. data> = an integer number according to the table on page 14-6.

Complies to standards: SCPI 1991.0, confirmed.

Device Status Register, Enable — command/query

Sets the enable bit of the Device Register 0.

Command syntax: **:STATus:DREGister0:ENABle**_<bit mask>

Where:

<bit mask> = an integer number according to the table below.

Bit No.	Weight	Condition
2	4	Enable monitoring of low limit
1	2	Enable monitoring of high limit
0	1	Enable monitoring of delay trig

Query syntax: :STATus:DREGister0:ENABle?

● *Returned format:* <bit mask>

Complies to standards: Philips Timer/Counter 1991.

Device Status Register, Event

Device Status Register, Event — query

Reads out the contents of the Device Event Register. Reading the Device Event Register clears the register.
See figure 14-11.

Query syntax: **:STATus:DREGister0:[EVENt]?**

- *Returned format:* <dec. data>

  Where:

  <Dec. data> = according to table below:

Bit No.	Weight	Condition
2	4	Last measurement below low limit.
1	2	Last measurement above high limit.
0	1	Delay trig, a trigger has been executed but not via the trigger control option.

Complies to standards: Philips Timer/Counter 1991.

Enable Device Status Reporting command

This command has a SCPI standardized effect on the status data structures. The purpose is to precondition these towards reporting only device-dependent status data.

- Only enable registers and transition filters are affected. Event and condition registers are not touched.
- The IEEE-488.2 enable registers, which are handled with the common commands ✱SRE and ✱ESE remain unchanged.
- All other enable registers and transition filters are set or cleared. Those relevant for this counter are:
  - All bits of the Device status Enable Registers are set to 1.
  - All bits of the Questionable Data Status Enable Registers and the Operation Status Enable Registers are set to 0.
- The following registers never change in the PM 6680, but they do conform to the standard :STATus:PRESet values.
  - All bits in the positive transition filters of Questionable Data and Operation status registers are 1.
  - All bits in the negative transition filters of Questionable Data and Operation status registers are 0.

Command syntax: **STATus:PRESet**

Complies to standards: SCPI 1991.0, confirmed.

Clear Status Command

Clear Status Command — command

The ✷CLS common command clears the status data structures by clearing all event registers and the error queue. It does not clear enable registers and transition filters. It clears any pending ✷WAI, ✷OPC, and ✷OPC?.

Command Syntax: ✷CLS

- *Example:* ✷CLS ↵

Complies to standards: IEEE 488.2 1987.

Standard Event Status Enable — command/query

Sets the enable bits of the standard event enable register. This enable register contains a mask value for the bits to be enabled in the standard event status register. A 1 in the enable register enables the corresponding bit in the status register.

An enabled bit will set the ESB (Event Status Bit) in the Status Byte Register if the enabled event occurs. See also status reporting on page 14-18.

Command Syntax: ✷ESE␣<dec. data>

Where:

<dec.data> = an integer number between 0 and 255 according to the table below:

Event Status Enable Register (1 = enable)		
Bit	**Weight**	**Enables**
7	128	PON, Power ON occurred
6	64	URQ, User Request
5	32	CME, Command Error
4	16	EXE, Execution Error
3	8	DDE, Device Dependent Error
2	4	QYE, Query Error
1	2	RQC, Request Control (not used)
0	1	Operation Complete

● *Example:* SEND→ ✷ESE␣36↵

In this example command error, bit 5, and query error, bit 2, will set the ESB-bit of the Status Byte if these errors occur.

Query Syntax: ✷ESE?

● *Returned Format:* <dec. data> ↵

Complies to standards: IEEE 488.2 1987.

Event Status Register

Event Status Register — query

Reads out the contents of the standard event status register. Reading the Standard Event Status Register clears the register.

Query Syntax: **✷ESR?**

- *Returned Format:* <dec. data> ↵

  Where:

  <dec. data> = an integer number between 0 and 255. See table on page 14-13.

Complies to standards: IEEE 488.2 1987.

Power-on Status Clear — command/query

Enables/disables automatic power-on clearing. The status registers listed below are cleared when the power on status flag is 1. When the flag is 0, the registers are unaffected by power on.

- Service request enable register (*SRE)
- Event status enable register (*ESE)
- Operation status enable register (:STAT:OPER:ENAB)
- Questionable data/signal enable register (:STAT:QUES :ENAB)
- Device enable registers (:STAT:DREG0:ENAB)
- This power on status clear flag is not affected by *RST.

Command Syntax: ***PSC**␣<decimal data>

Where:

<dec. data> = a number which rounds to 0 turns off automatic power on clearing. Any other value turns it on.

● *Example:* *PSC␣1↵

This example enables automatic power on clearing.

Query Syntax: ***PSC?**

● *Returned Format:* 1 | 0 ↵

Where:

1 is enabled and 0 is disabled.

Complies to standards: IEEE 488.2 1987.

Service Request Enable

Service Request Enable — command/query

The Service Request Enable command sets the service request enable register bits. This enable register contains a mask value for the bits to be enabled in the status byte register. A 1 in the enable register enables the corresponding bit in the status byte register to generate a Service Request.

Command Syntax: ✷SRE␣<dec. data>

Where:

<dec.data> = the sum (between 0 and 255) of all bits that are set. See table below:

Service Request Enable Register (1 = enable)		
Bit	**Weight**	**Enables**
7	128	OPR, Operation Status
6	64	RQS, Request Service
5	32	ESB, Event Status Bit
4	16	MAV, Message Available
3	8	QUE, Questionable Data/Signal Status
2	4	EAV, Error Available
1	2	Not used
0	1	Device Status

● *Example:* ✷SRE␣<16>↵

In this example a service request is generated when a message is available in the output queue.

Query Syntax: ✷SRE?

Returned Format:

<integer>↵

Where:

<integer> = the sum of all bits that are set.

Complies to standards: IEEE 488.2 1987.

Status Byte Query

query

Reads out the value of the Status Byte. The Master Summary Status bit (MSS) is reported on bit 6, not the Request Service (RQS). The MSS is set if the instrument has one or more reasons for requesting service.

Query Syntax:

*STB?

- *Returned Format:* <integer>↵

Where:

<integer> = the sum (between 0 and 255) of all bits that are set. See table below:

Status Byte Register (1 = true)			
Bit	**Weight**	**Name**	**Condition**
7	128	OPR	Enabled operation status has occurred.
6	64	MSS	Reason for requesting service
5	32	ESB	Enabled status event condition has occurred
4	16	MAV	An output message is ready
3	8	QUE	The quality of the output signal is questionable
2	4	EAV	Error available
1	2		Not used
0	1	DREG0	Enabled status device event conditions have occurred

See also:

If you want to read the status byte with the RQS bit, use serial poll.

Complies to standards: IEEE 488.2 1987.

This page is intentionally left blank.

How to use Status Reporting

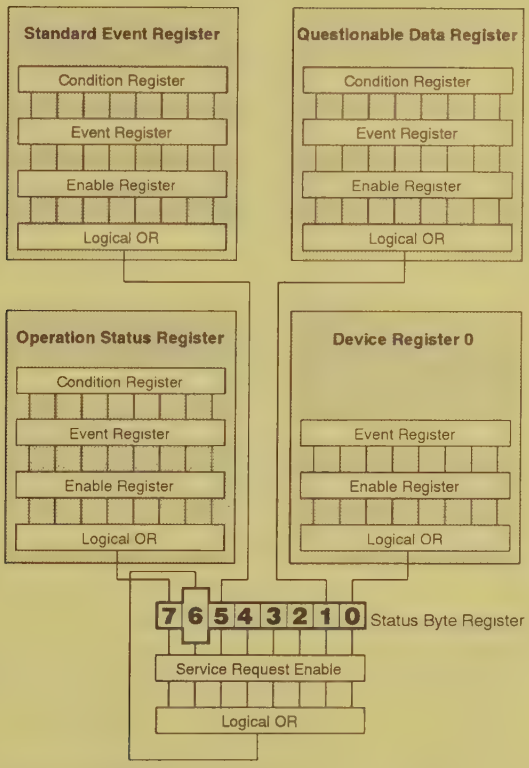


Fig. 14-3 *PM 6680 Status register structure.*

- *Status reporting model*

This section handles the status reporting features of the timer/counter. Figure 14-4 shows a block diagram for the IEEE 488.2 Standard Status data structure. The status model is based on reporting events or queue status. An event is a change of a device condition. The events and the device conditions may be represented in registers (event

and condition registers), but may also be logical states of the hardware. An example of an event is when an error occurs in the device.

You can select which events in the event registers the instrument should summarize (logical OR) into the IEEE 488.1 status byte register, by programming the event enable register. The instrument summarizes the events from the standard event status register in the ESB (event status) bit (bit 5) and the events from a device defined event status register in one of the bits 0, 1, 2, 3, or 7 (see fig. 14-4).

When you read the event registers by using the related query, you will also clear the registers you read. You can also clear them with the ✱CLS (Clear Status) command, which clears all event registers.

You can clear an event enable register by programming it with a value of zero. Power-on clears all event enable registers and the service request enable register if the power-on status clear flag is set TRUE (see the common command ✱PSC).

The not-empty status of the output queue is reflected in the message available (MAV) bit (bit 4).

You can select which status byte bits will cause the device to generate a service request (setting the RQS-bit TRUE) by setting bits in the service request enable register.

You can read the status byte using a serial poll or the ✱STB? query. This will not affect the contents of the status byte (except the RQS bit when read by serial poll).

The instrument only clears the summary message bit, (standard or device defined) in the status byte, when you clear or read the related event register, or if a queue model status data structure is tied to the summary bit when the queue is empty.

Example: The instrument clears the MAV bit in the status byte register when the output queue is empty, i.e., when it has sent the complete response message.

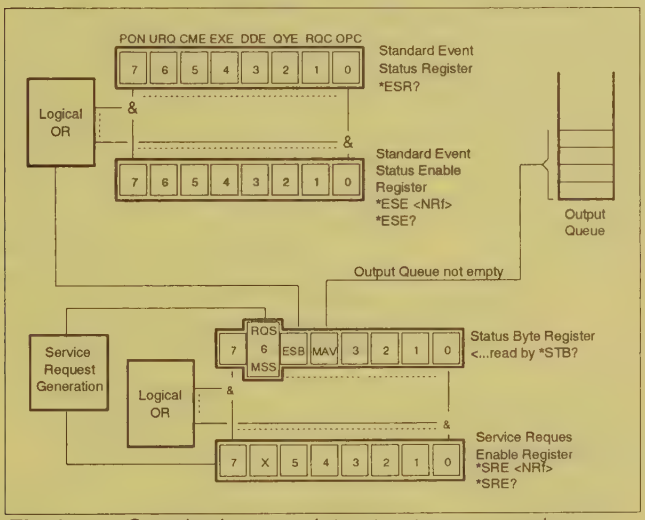

PON URQ CME EXE DDE QYE RQC OPC
7 6 5 4 3 2 1 0
Standard Event Status Register *ESR?
Logical OR
&
&
Standard Event Status Enable Register *ESE <NRf> *ESE?
Output Queue
Output Queue not empty
Service Request Generation
RQS
7 6 MSS ESB MAV 3 2 1 0
Status Byte Register <...read by *STB?
Logical OR
&
&
7 X 5 4 3 2 1 0
Service Reques Enable Register *SRE <NRf> *SRE?

Fig 14-4 *Standard status data structures, overview.*

All bits (except bit 6, RQS) in the status byte register are bits representing summary messages. Each bit summarizes an overlying status data structure. An example of a register model overlying status data structure is given in figure 14-5.

Summary bits always track the status of the associated status data structure. Reading the status byte register by serial poll or by ✱STB? does not clear its summary bits. The instrument clears a summary bit only when it clears the associated event register or queue (see above).

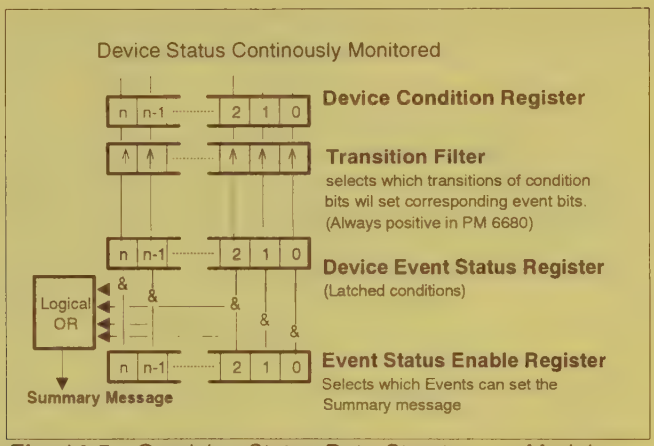


Fig. 14-5 Overlying Status Data Structure — Model.

- *Service request and reading of the status byte*

Selecting a summary message to generate SRQ

You can program the service request enable register (*SRE <number>) to select which summary message(s) in the status byte register will cause the device to generate a service request and as a result, set the RQS (requested service) bit in the same register. When a service request occurs, the SRQ-line of the GPIB will be activated. Whether the controller will react on the service request or not depends on the controller program. The controller may be interrupted on occurrence of a service request or it may regularly test the SRQ-line, or the controller may not react at all.

Example: `*SRE 16`

This sets bit 4 ($16=2^4$) in the service request enable register (see figure 14-4). The instrument AND:s this bit with the corresponding bit (bit 4 = MAV) in the status byte register. It directs the result to the input of a logical OR, the output of which sets the MSS (Master Summary Status) bit. Now

when a response message is available in the output queue, it sets the MAV bit TRUE. As a consequence, it sets the MSS bit TRUE and generates a service request by setting the RQS (requested service) bit TRUE. When the GPIB controller reacts to the service request with a serial poll, the instrument resets the RQS bit (FALSE). When a new service request occurs, the instrument sets the RQS-bit TRUE again. The MSS bit remains TRUE after the serial poll. When the GPIB controller reads the response message from the device, the instrument empties the output queue and resets the MAV-bit. This also resets the MSS bit, if MAV was the only enabled summary message.

You can read the status byte register in two ways:

- Using the Serial Poll (IEEE-488.1 defined).
  Response:
  Bit 6: RQS message, shows a service request.
  Other bits: Summary messages (same as using *STB?)
  A serial poll sets the RQS bit FALSE, but does not change other bits.
- Using the common query *STB?
  Response:
  Bit 6: MSS message, shows that there is a reason for service request.
  Other bits: same as using serial poll.
  Reading the response will not alter the status byte.

The text above does not completely explain the difference between the MSS and RQS bit. An IEEE 488.2 defined synchronization mechanism takes care of not losing a new reason for service request, while a serial poll is in progress. That is, during a serial poll, the instrument may not yet have removed the current service request when a new reason for service request occurs. It sets the MSS bit TRUE, but it holds off the new service request until it has finished the current serial poll. The instrument now carries out the service request and sets the RQS bit TRUE. Now the controller can react to the new service request.

Standard event status register

- *Bit definitions*

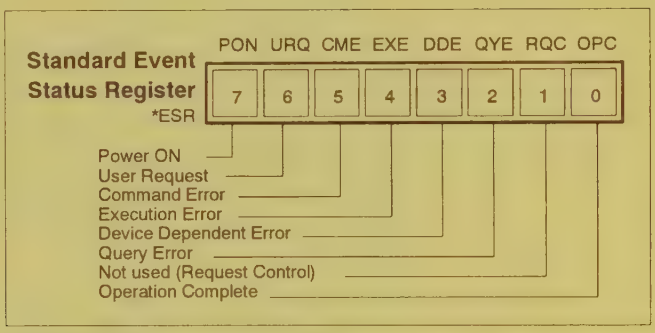


Fig. 14-6 *Bits in the standard event status register.*

Bit 7 - Power on (PON)

Shows that the device's power supply has been turned off and on (since the last time the controller read or cleared this register).

Bit 6 - User request (URQ)

Shows that the user has activated the URQ signal. This bit will be set regardless of the remote local state of the device. The purpose of this signal is, for example, to call for the attention of the controller by generating a service request. (How the URQ signal will be activated is device dependent. In PM 6680 all active keys will set the URQ signal, except reset/local).

Bit 5 - Command error (CME)

Shows that the instrument has detected a command error. This means that the device has received data that violates the syntax rules for program messages.

Bit 4 - Execution error (EXE)

Shows that the device detected an error while trying to execute a command (see 'Error handling' on page 21-19). The command is syntactically correct, but the instrument cannot execute it, for example because a parameter is out of range.

Bit 3 - Device-dependent error (DDE)

A device-dependent error is any device operation that did not execute properly because of some internal condition, for instance overload. This bit shows that the error was not a command, query or an execution error.

Bit 2 - Query error (QYE)

The output queue control detects query errors. For example the QYE bit shows the unterminated, interrupted, and deadlock conditions. For more details, see 'Error handling' on page 21-19.)

Bit 1 - Request control (RQC)

Shows to the controller that the device wants to become the active controller-in-charge. Not used in the PM 6680.

Bit 0 - Operation complete (OPC)

The instrument only sets this bit TRUE in response to the operation complete command (*OPC). It shows that the device has completed all previously started actions.

● *Using the standard event status register*

Reading: ✷ESR? This also clears the register.

Example of response: 136

Explanation:
$136 = 2^7 + 2^3$
bit 7 = PON = Power-on
bit 3 = DDE = Device Dependent Error

Writing: Not possible

Clearing:

1. ✷CLS
   CAUTION: Not only clears this register but also clears all status data structures including the output queue.

2. ✷ESR?

Device defined status registers

The standard and the device-defined status data structures can be registers or queues. IEEE-488.2 defines the standard status data structures. The standard status data structure #4 is a queue model structure (output queue), and the standard status data structure #5 is a register model structure (event status). They reflect their summary message in the MAV and ESB bits of the status byte register respectively.

- PM 6680 has two SCPI data structures and one device defined status structure:
  Data structure #7 The "operation status group" summarized in bit 7, OPR. This is a SCPI defined group.
  Data structure #3 The "questionable data/signal group" summarized in bit 3, QUE. This is a SCPI defined group.
  Data structure #0 The "Device status group" summarized in bit 0, DREG0.

These groups are 16-bits wide while the status byte and the standard status groups are 8-bits wide.

- *Device defined status data structure*

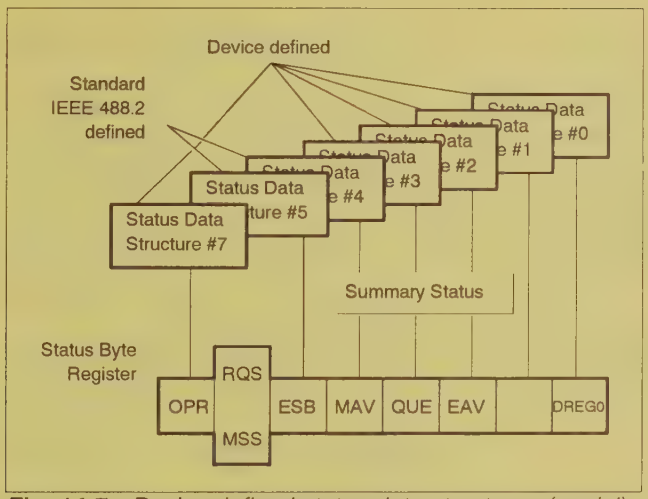
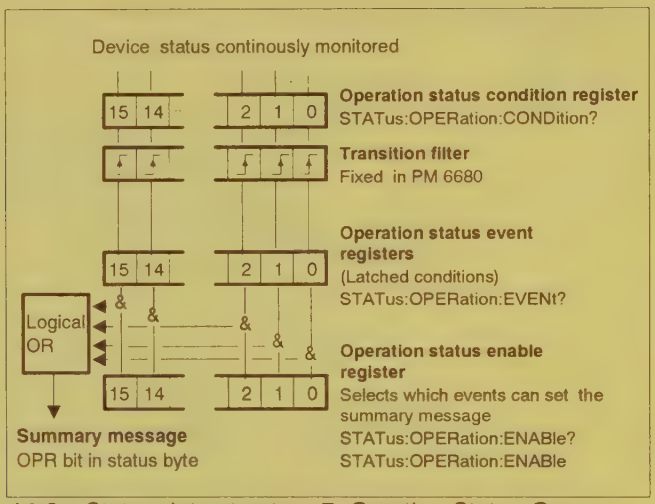


Fig. 14-7 *Device-defined status data structures (model)*

The model shown above is the same as the model used for the standard event status, figure 14-4. The register model may include transition filters (see figure 14-5) for the creation of events derived from the device-defined conditions.



14-8 *Status data structure 7, Oeration Status Group.*

PM 6680 status condition registers

The condition registers continuously monitors the hardware and software status of the device. The registers are updated in real time and they can only be read. The operation status condition register is read with STATus:OPERation:CONDition? The questionable data/signal condition register is read with STATus:QUEStionable:CONDition?
The Device Register has no condition register.

The condition registers cannot be read during a changing condition. So, use SRQ to monitor conditions.

PM 6680 transition filters

The transition filter specifies if the beginning or the ending of a condition should set the corresponding bit in the event status register. The transition filters in PM 6680 are fixed to always detect a positive transition (False to True).

PM6680 status event registers

These registers latches the events from the condition registers. The bits in these registers stay set until the register is read or cleared with the *CLS command (clear status). *CLS clears all event status registers, both the device defined and the standard defined. It is neither meaningful nor permitted to write to an event status register. The registers are read by:

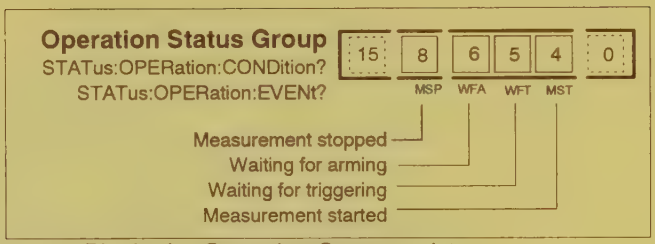
- STATus:OPERation:EVENt? Operation status, event.
- STATus:QUEStionable:EVENt? Questionable status, event.
- STATus:DREGister0:EVENt? Warning & limit monitoring events.

PM 6680 status event enable registers

- The enable register selects which bits of the event status registers should be summarized in the status byte. You can clear an event status enable register by programming it with a value of zero or by sending *CLS. These enable registers are set by:
- STATus:OPERation:ENABle␣<bit mask> Operation status, enable.
- STATus:QUEStionable:ENABle␣<bit mask> Questionable status, enable.
- STATus:DREGister0:ENABle␣<bit mask> Warning & limit monitoring enable.

NOTE: If a condition is to be summarized in the status byte, the corresponding bit must be selected in the enable register.

● *Operation status group*



14-9 *Bits in the Operation Status register.*

Bit 8 - Measurement stopped (MSP)

This bit indicates that the counter is not measuring. It is set when the measurement, or sequence of measurements, stops.

Bit 6 - Wait for bus arming (WFA)

This bit indicates that the Timer/Counter is waiting for bus arming in the arm state of the trigger model.

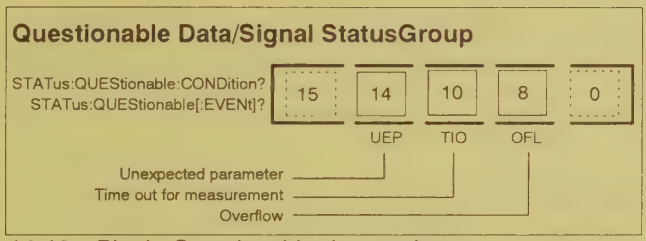
Bit 5 - Waiting for trigger and/or external arming (WFT)

This bit is set when the counter is ready to start a new measurement via the trigger control option (488.2), for the shortest possible trigger delay. The counter is now in the wait for trigger state of the trigger model.

Bit 4 - Measurement started (MST)

This bit indicates that the counter is measuring. It is set when the measurement, or sequence of measurements, start.

● *Questionable data/signal status group*



14-10 *Bits in Questionable data register.*

Bit 14 - Unexpected parameter (UEP)

This bit is set if a parameter that is valid according to SCPI is received but it cannot be executed by PM 6680.

Bit 10 - Time out for measurement (TIO)

This bit is set when the measurement has been abandoned because the internal time out has elapsed.
See also :SYST:TOUT.

Bit 8 - Overflow (OFL)

This bit is set when the measurement did not complete due to overflow.

● *Device register 0*

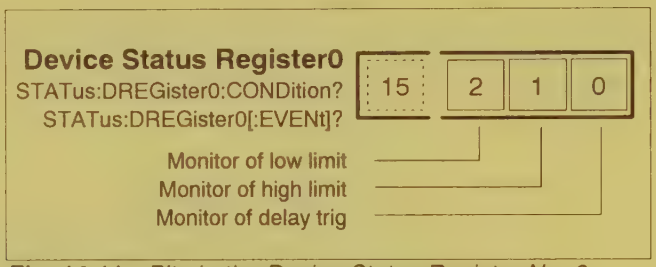


Fig. 14-11 *Bits in the Device Status Register No. 0. :STATus:DREGister0? reads out the contents of the Device Startus event Register 0 and clears the register.*

Bit 2 - Monitor of low limit

This bit is set when the low limit passed from above

Bit 1 - Monitor of high limit

This bit is set when the high limit passed from below

Bit 0 - Monitor of delay trig

This bit is set when a trigger has been executed but not via the trigger control option.

● *Enable all Status Reporting*

Sets all device status enable registers(DREG0) to 1, that is enabled.

STATus:PRESet

● *Queue status data structure*

The queue model status data structure allows reporting sequential status. Only the standard output and the error queue in the PM 6680 uses this data structure. The output queue with its associated MAV bit is an example of a queue model status data structure, although this is a standard status data structure (see figure 14-24).

Status of the output queue

The MAV (message available) summary-message is a standard IEEE-488.2 defined message, which appears in bit 4 of the status byte register. A '1' in bit 4 (MAV summary-message TRUE), shows that the instrument has data in the output queue. When the controller reads the data, the output queue will be emptied. For as long as the output queue holds one or more data bytes, the MAV bit in the status byte register remains TRUE. When all the data bytes have been sent, the output queue is empty and MAV is set FALSE.

Status of the error message queue

The EAV (error message available) summary message is a message appearing in bit 2 of the status byte register. A '1' in bit 2 shows that the instrument has error messages in its error message queue. The messages may be read by the :SYSTem:ERRor? query. When a message has been read it will be removed from the queue. When the error message queue is empty bit 2 in the status byte register will be set to '0' (False).

This page is intentionally left blank.

Chapter 15

Synchronization

Operation Complete

Operation Complete **command**

The Operation Complete command causes the device to generate the operation complete message in the Standard Event Status Register when all pending selected device operations have been finished.

Command Syntax: ✱OPC

Complies to standards: IEEE 488.2 1987.

Operation Complete Query

query

Operation Complete query. The Operation Complete query places an ASCII character 1 into the device's Output Queue when all pending selected device operations have been finished.

Query Syntax: ***OPC?**

- *Returned Format:* 1↵

Complies to standards: IEEE 488.2 1987.

Wait-to-continue command

The Wait-to-Continue command prevents the device from executing any further commands or queries until execution of all previous commands or queries have been completed.

Command Syntax: ✷WAI

● *Example:* SEND→ `:MEAS:FREQ?;✷WAI;MEAS:PTP?`↵

In this example ✷WAI makes the instrument perform both the frequency and the peak to peak voltage measurement. Without ✷WAI, only the voltage measurement would be performed.

READ← `146E6;1.22`↵

Complies to standards: IEEE 488.2 1987.

Chapter 16

System Subsystem

System Subsystem Command Tree

<HEADER>	Parameters
:SYSTem	
:COMMunicate	
:GPIB	
:ADDRess	<numeric value>|MIN|MAX
:ERRor?	
:KEY	<numeric value>|MIN|MAX
:PRESet	
:SET	<block data>
:TOUT	
[:STATe]	<boolean>
:TIME	<time out value>
:UNPRotect	
:VERSion?	

*IDN?
*OPT?
*PUD
*RST

GPIB Address — command/query

This command sets the GPIB address. This selection overrides the switches on the rear panel of the counter. The set address is valid until a new address is set, or until the counter is switched off (set to standby).

Command syntax: **:SYSTem:COMMunicate:GPIB:ADDRess**␣ <numeric value>|MAX|MIN

Where:

<numeric value> is a number between 0 and 30.
MIN sets address 0.
MAX sets address 30.

Query syntax: :SYSTem:COMMunicate:GPIB:ADDRess?

● *Returned format:>* <numeric value>↵

Complies to standards: SCPI 1991.0, confirmed.

Error Description

Error Description — query

Queries for an ASCII text description of an error that occurred. The error messages are placed in an error queue, with a FIFO - First In-First Out structure. This queue is summarized in an Error AVailable (EAV) bit in the status byte.

Query syntax: **:SYSTem:ERRor?**

- *Returned format:* <error number>,"<string>"↵

Where:
<string> = an error description as ASCII text.

See also: Chapter 21, error messages.

Complies to standards: SCPI 1991.0, confirmed.

Key command

This command is used in testing and simulates a press on the specified key. The code is the same as the hardware code for each key. KEY 0 means release key. KEY 0 must be used, otherwise the counter may hang.

Command syntax: **:SYSTem:KEY** <decimal value>|MIN|MAX
Where: <decimal value> =Code according to table below.

Key	Code	Key	Code	Key	Code
FILTER A:	42	IMPedance A:	43	SLOPE A:	44
ATTenuator A:	47	COUPLing A:	46	SET TRIG LEV A:	45
TRIG LEV AUTO:	57	SET TRIG LEV B:	58	SWAP AB:	49
IMPedance B:	51	SLOPE B:	50	ATTenuator B:	60
COUPLing B:	59	COMmon B via A:	52	CHECK:	53
TOT ST/STOP:	61	HOLD OFF ON:	54	HOLD OFF SET:	62
FUNCtion right:	34	FUNCtion left:	33	AUX MENU:	39
MENU:	40	MEAS TIME SET:	35	MEAS TIME DISPL HOLD:	36
MEAS TIME SINGLE:	38	MEAS TIME RESTART:	37	ARM START:	18
ARM STOP:	26	PROCESS MATH:	17	PROCESS STAT:	25
PROCESS K=:	19	PROCESS L=:	20	PROCESS M=:	21
DATA ENTRY keys					
SAVE/CLEAR:	22	RECALL/EE:	23	ENTER:	24
0:	16	. (point):	15	+/-:	14
1:	09	2:	10	3:	11
4:	01	5:	02	6:	03
7:	08	8:	07	9:	06
X n-1:	05	X	04	SELECT/SET UP:	12
SELECT/SET DOWN:	13				

● *Example:* SEND→ `:SYST:KEY 42;:SYST:KEY 0`

In this example the filter key for input A is pressed and released.

Complies to standards: SCPI 1991.0, confirmed.

Preset

Preset command

This command sets the counter to the same state as when the front panel key LOCAL/PRESET is pressed in local mode.

NOTE: These are not exactly the same settings as after ✱RST.

Command syntax: **:SYSTem:PRESet**

Complies to standards: SCPI 1991.0, confirmed.

Read or Send Settings — command/query

Transmits in binary form the complete current state of the instrument. This data can be sent to the instrument to later restore this setting. This command has the same function as the ✱LRN? common command with the exception that it returns the data only as response to :SYST:SET?. The query form of this command returns a definite block data element.

Command syntax: **:SYSTem:SET**␣<block data>

Where:

<block data> is the instrument setting previously retrieved via the :SYSTem:SET? query.

Query syntax: :SYSTem:SET?

● *Returned format:* <block data>↵

Where:

<block data> is #292<92 data bytes>

SEND→ :SYST:SET?

READ← :SYST:SET␣#2924RH.+..A....d..A..d..+....ABD.....␣?....
.....␣?.....␣?.....d.d.A.A.M

Complies to standards: SCPI 1991.0, confirmed.

Timeout ON/OFF

Timeout ON/OFF — command/query

This command switches on or off the timeout.

When time out is enabled the measurement attempt will be abandoned when the time set with :SYST:TOUT:TIME has elapsed. A zero result will be sent to the controller instead of a measurement result and the time out bit in the :STATus:QUEStionable register will be set.

Command syntax: **:SYSTem:TOUT[:STATe]**␣<boolean>

Query syntax: :SYSTem:TOUT[:STATe]?

● *Returned format:* 0|1↵

Where:

0 means no timeout

1 means that the timeout set by :SYSTem:TOUT:TIME is used.

✷RST condition: 0

Complies to standards: Philips Timer/Counter 1991.

Timeout, Set — command/query

This command sets the timeout in seconds. The range is 0 to 25.5 s. Note that you must enable timeout using :SYST:TOUT␣ON for this setting to take effect.

Command syntax: **:SYSTem:TOUT:TIME**␣<numeric value>|min|max

Query syntax: :SYSTem:TOUT:TIME?

● *Returned format:* <numeric value>↵

****RST condition:*** 0.1

Complies to standards: SCPI 1991.0, confirmed.

Unprotect

Unprotect command

This command will unprotect the user data (set/read by ✷PUD) and front setting memories 10-19 until the next PMT (Program message terminator) or Device clear or Reset (✷RST). This makes it necessary to send an unprotect command in the same message as for instance ✷PUD.

Command syntax: :SYSTem:UNPRotect

● *Example* Send → `:SYST:UNPR;✷PUD␣#240Calibrated␣1992-04-01,␣inventory␣No.1234`

Where:

means that <arbitrary block program data> will follow.
2 means that the two following digits will specify the length of the data block
40 is the number of characters in this example

Complies to standards: Philips Timer/Counter 1991.

System Version — query

This query returns the SCPI system version that this instrument complies to.

Query syntax: **:SYSTem:VERSion?**

- *Returned format:* <year>,<revision>

  Where <year> is the year of publication of the complied standard and <revision> is the number of the SCPI standard.

Complies to standards: SCPI 1991.0, confirmed.

Protected User Data

Protected User Data — command/query

Protected user data. This is a data area in which the user may write any data up to 64 characters. The data can always be read, but you can only write data after unprotecting the data area. A typical use would be to hold calibration information, usage time, inventory control numbers, etc.

Command syntax: ✷PUD␣<arbitrary block program data>

The content at delivery is:

#232 FACTORY CALIBRATED : 19YY-MM-DD

- Where:
  YY = year
  MM = month
  DD = day

● *Example* Send → `:SYST:UNPR;✷PUD␣#240Calibrated␣1992-04-01,␣inventory␣No.1234`

Where:
means that <arbitrary block program data> will follow.
2 means that the two following digits will specify the length of the data block.
40 is the number of characters in this example.

Query syntax: ✷PUD?

● *Returned format:* <arbitrary block response data>↵

Where:
<arbitrary block program data> is the data last programmed with ✷PUD.

Complies to standards: IEEE 488.2 1987.

Reset command

The Reset command resets the timer/counter. It is the third level of reset in a 3-level reset strategy, and it primarily affects the timer/counter functions, not the IEEE 488 bus.

The timer/counter settings will be set to the default settings listed on page 2-6. All previous commands are discarded, macros are disabled, and the timer/counter is prepared to start new operations.

Command Syntax: ✱RST

- *Example:* ✱RST ↵

See also: Default settings on page 2-6.

Complies to standards: IEEE 488.2 1987.

Chapter 17

Test Subsystem

Test Subsystem Command Tree

Test Subsystem Command Tree

Header	parameters
:TEST	
├→ :CHECk	␣<boolean>
└→ :SELect	␣RAM|ROM|LOGic|ALL

✱TST

Select Check signal — command/query

This command connects the internal 10 MHz reference signal to the measuring logic, instead of an external measuring signal. This makes it possible to test all functions but Volt and Rise/Fall time.

Command syntax: **:TEST:CHECk**␣<boolean>

Where <boolean> = 1|0
1 means test signal connected and 0 means test signal disconnected.

NOTE: Selecting channel 6 when entering measuring channel for :CONFigure :MEASure etc. also selects the 10 MHz reference.

Query syntax: **:TEST:CHECk?**

● *Returned format:* 1|0↵

✷RST condition: 0

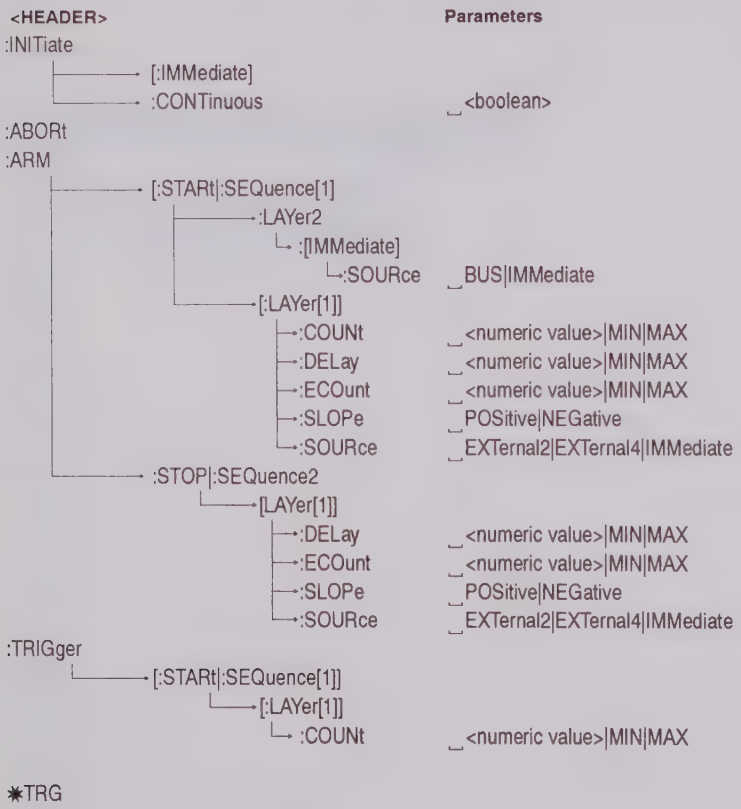
Complies to standards: Philips Timer/Counter 1991.

This page is intentionally left blank.

Chapter 18

Trigger/Arming Subsystem

Trigger Subsystem Command Tree



Initiate Measurement command

The INITiate command initiates a measurement. Executing an INITiate command changes the counter's trigger subsystem state from "idle-state" to "wait-for-bus-arm-state" (see fig. 18-1). The trigger subsystem will continue to the other states depending on programming. With the ✱RST setting, the trigger subsystem will bypass all its states and make a measurement, then return to idle state. See also 'How to use the Trigger Subsystem' at the end of this chapter.

Command syntax: **:INITiate[:IMMediate]**

Complies to standards: SCPI 1991.0, confirmed.

Continuously Initiated — command/query

With this command the trigger system could continuously be initiated. When Continuous is OFF the trigger system remains in the "idle-state" until Continuous is set to ON, or the :INITiate [:IMMediate] is received. When Continuous is set to ON the completion of a measurement cycle immediately starts a new trigger cycle without entering the "idle-state", i.e., the counter is continuously measuring and storing response data.

Command syntax: **:INITiate:CONTinuous** <Boolean>

Query syntax: :INITiate:Continuous?

● *Returned format:* <Boolean>↵

****RST condition:*** OFF

Complies to standards: SCPI 1991.0, confirmed.

Abort Measurement — command

The ABORt command terminates a measurement. The trigger subsystem state is set to "idle-state".

Command syntax: :ABORt

Type of command: Aborts all previous measurements if *WAI is not used.

Complies to standards: SCPI 1991.0, confirmed.

Bus Arming Exit **command**

This command overrides the waiting for bus arm, provided the source is set to bus. When this command is issued the counter will immediately exit the "wait-for-bus-arm" state.

The counter generates an error if it receives this command when the trigger subsystem is not in the "wait-for-bus-arm" state.

If the Arming source is set to Immediate, this command is ignored.

Command syntax: **:ARM[:STARt|:SEQuence[1]]:LAYer2:[IMMediate]**

SEND→ `:ARM:LAY2`↵

Complies to standards: SCPI 1991.0, confirmed.

Bus Arming on/off — command/query

Switches between Bus and Immediate mode for the "wait-for-bus-arm" function, (layer 2). GET and ✷TRG triggers the timer/counter if Bus is selected as source.

If the counter receives GET/✷TRG when not in "wait-for-bus-arm" state, it ignores the trigger and generates an error.

It also generates an error if it receives GET/✷TRG and bus arming is switched off (set to IMMediate).

Command syntax: **:ARM[:STARt|:SEQuence[1]]:LAYer2 :SOURce** ␣BUS|IMMediate

SEND→ `:ARM:LAY2:SOUR␣BUS↵`

Query syntax: :ARM[:STARt|:SEQuence[1]]:LAYer2:SOURce?

● *Returned format:* BUS|IMM↵

Complies to standards: SCPI 1991.0, confirmed.

No. of Measurements on each Bus arm

No. of Measurements on each Bus arm — command/query

This count variable controls the upward exit of the "wait-for-bus-arm" state (ARM:STARt:LAY1). The counter loops the trigger subsystem downwards COUNt number of times before it exits to the idle state.

This means that a COUNt No. of measurements can be done for each Bus arming or INITiate.

NOTE: The actual number of measurements made on each INIT equals to:
(:ARM:START:COUN)*(:TRIG:START:COUNT)

Range: 1 to 65536.

Command syntax: **:ARM[:STARt|:SEQuence[1]][:LAYer[1]] :COUNt**
␣<numeric value>|MIN|MAX

Where:
<numeric value> is a number between 1 and 65536.
1 switches the function OFF.

SEND→ `:ARM:COUN␣100↵`

Query syntax: :ARM[:STARt|:SEQuence[1]][:LAYer[1]]:COUNt?

● *Returned format:* <numeric value>↵

✷RST condition: 1

Complies to standards: SCPI 1991.0, confirmed.

Delay after External Start Arming — command/query

This command sets a delay between the pulse on the arm input and when the counter starts measuring. Start delay cannot be used at the same time as stop delay (:ARM:STOP:DEL). The delay is only active when the following is selected:
:ARM:STARt:SOURce␣EXT2| EXT4.

Only one of the delays: ARM STAR:DEL, ARM STOP:DEL, ARM:STAR:ECO and ARM:STOP:ECO can be used at a time. If you program one delay, the other three delays will be reset to their ✷RST values.

Range: 200 ns to 1.6 s.

Command syntax: **:ARM[:STARt|:SEQuence[1]][:LAYer[1]] :DELay**
␣<numeric value>|MIN|MAX

Where:
<numeric value> is a number between $200*10^{-9}$ and 1.6.
0 switches the delay OFF.

SEND→ `:ARM:DEL␣0.1↵`

Query syntax: :ARM[:STARt|:SEQuence[1]][:LAYer[1]]:DELay?

● *Returned format:* <numeric value>↵

✷RST condition: 0

Complies to standards: SCPI 1991.0, confirmed.

External Events before Start Arming — command/query

This command sets the number of negative edges required on the B-input (EXT2) before the counter starts measuring (Start Arming Delay by events). Start Arming delay by events cannot be used at the same time as stop Arming delay by events (:ARM:STOP :ECO).

NOTE: The delay is only active when :ARM:START:SOUR␣ EXT2|EXT4 is selected.

Range: 2 to 16 777 215.

Only one of the delays: ARM STAR:DEL, ARM STOP:DEL, ARM:STAR:ECO, and ARM:STOP:ECO can be used at a time. When you program this delay, the other three delays will be reset to their ✷RST values.

Command syntax **:ARM[:STARt|:SEQuence[1]][:LAYer[1]] :ECOunt**
␣<numeric value>|MIN|MAX

Where:
<numeric value> is a number between 2 and 16 777 215.
1 switches the delay by events OFF.

SEND→ `:ARM:ECO␣25↵`

Query syntax: :ARM[:STARt|:SEQuence[1]][:LAYer[1]]:ECOunt?

● *Returned format:* <numeric value>↵

✷RST condition: 1

Complies to standards: SCPI 1991.0, confirmed.

External Arming Start Slope — command/query

Sets the slope for the start arming condition.

Command syntax: **:ARM[:STARt|:SEQuence[1]][:LAYer[1]]:SLOPe _POSitive|NEGative**

SEND→ **:ARM:SLOP _NEG↵**

Query syntax: :ARM[:STARt|:SEQuence[1]][:LAYer[1]]:SLOPe?

● *Returned format:* POS|NEG↵

✷RST condition: POS

Complies to standards: SCPI 1991.0, confirmed.

External Arming Start Source

External Arming Start Source — command/query

Selects channel 2 (input B) or channel 4 (Input E) as arming input, or switches off the start arming function. When switched off, both the ECOunt and the DELay are inactive.

Command syntax: **:ARM[:STARt|:SEQuence[1]][:LAYer[1]] :SOURce**
␣EXTernal2|EXTernal4|IMMediate

Where:
EXTernal2 is input B
EXTernal4 is input E
IMMediate is Start arming OFF

SEND→ `:ARM:SOUR␣EXT4↵`

Query syntax: :ARM[:STARt:SEQuence[1]]:LAYer1:SOURce?

● *Returned format:* EXT2|EXT4|IMM↵

✷RST condition: IMM

Complies to standards: SCPI 1991.0, confirmed.

No. of Triggerings on each Ext Arm start — command

Sets how many measurements the instrument should make for each ARM:STARt condition, (block arming).

These measurements are done without any additional arming conditions before the measurement. This also means that stop arming is disabled for the measurements inside a block.

NOTE: The actual number of measurements made on each INIT equals to:
(:ARM:START:COUN)*(:TRIG:START:COUNT)

Command syntax: **:TRIGger[:STARt|:SEQuence[1]][:LAYer[1]]:COUNt** _<numeric value>|MIN|MAX

Where:
<numeric value> is a number between 1 and 2500.

SEND→ `:TRIG:COUN 50`↵

****RST condition:*** 1

Complies to standards: SCPI 1991.0, confirmed.

Delay after External Stop Arming

Delay after External Stop Arming — command/query

This command sets a delay between the stop condition on the arm input and when the counter stop is armed. Stop delay cannot be used at the same time as start delay (:ARM :STAR:LAY2:DEL).

Range: 200 ns to 1.6 s.

Only one of the delays: ARM STAR:DEL, ARM STOP:DEL, ARM:STAR:ECO and ARM:STOP:ECO can be used at a time. If you program one delay, the other three delays will be reset to their ✱RST values.

NOTE: The delay is only active when :ARM:START:SOUR␣ EXT2|EXT4 is selected.

Command syntax: **:ARM:STOP|:SEQuence2[:LAYer[1]]:DELay**
␣<numeric value>|MIN|MAX

Where:
<numeric value> is a number between $200*10^{-9}$ and 1.6.
0 switches the delay OFF.

SEND→ `:ARM:STOP:DEL␣500E-9↵`

Query syntax: :ARM:STOP|:SEQuence2[:LAYer[1]]:DELay?

● *Returned format:* <numeric value>↵

✱RST condition: 0

Complies to standards: SCPI 1991.0, confirmed.

External Events before Stop Arming — command/query

This command sets the number of negative edges required on the B-input (EXT2) before the counter stops measuring (Stop Arming delay by events). Stop Arming delay by events cannot be used at the same time as start Arming delay by events (:ARM:STAR :LAY2:ECO).

Range: 2 to 16 777 215.

Only one of the delays: ARM STAR:DEL, ARM STOP:DEL, ARM:STAR:ECO and ARM:STOP:ECO can be used at a time. When you program one delay, the other three delays will be reset to their ✷RST values.

NOTE: The delay is only active when :ARM:START:SOUR␣ EXT2|EXT4 is selected.

Command syntax: **:ARM:STOP|:SEQuence2[:LAYer[1]]:ECOunt**
␣<numeric value>|MIN|MAX

Where:
<numeric value> is a number between 2 and 16 777 215.
1 switches the delay by events OFF.

SEND→ `:ARM:STOP:ECO␣200↵`

Query syntax: :ARM:STOP|:SEQuence2[:LAYer[1]]:ECOunt?

● *Returned format:* <numeric value>↵

✷RST condition: 1

Complies to standards: SCPI 1991.0, confirmed.

External Stop Arming Slope

External Stop Arming Slope — command/query

Sets the slope for the stop arming condition.

Command syntax: **:ARM:STOP|:SEQuence2[:LAYer[1]]:SLOPe**
␣POSitive|NEGative

SEND→ **:ARM:STOP:SLOP␣NEG↵**

Query syntax: :ARM:STOP|:SEQuence2[:LAYer[1]]:SLOPe?

- *Returned format:* POS|NEG↵

***RST condition:** POS

Complies to standards: SCPI 1991.0, confirmed.

External Stop Arming Source — command/query

Selects channel 2 (input B) or channel 4 (Input E) as stop arming input, or switches off the stop arming function. When switched off both ECOunt and DELay are inactive.

Command syntax: **:ARM:STOP|:SEQuence2[:LAYer[1]] :SOURce**
␣EXTernal2|EXTernal4|IMMediate.

Where:
EXTernal2 is input B
EXTernal4 is input E
IMMediate is Stop arming OFF

SEND→ **:ARM:STOP:SOUR␣EXT4↵**

Query syntax: :ARM:STOP|:SEQuence2[:LAYer[1]]:SOURce?

● *Returned format:* EXT2|EXT4|IMM↵

✷RST condition: IMM

Complies to standards: SCPI 1991.0, confirmed.

Trigger

Trigger — command

The Trigger command is the device-specific equivalent of the IEEE 488.1 defined Group Execute Trigger, and has exactly the same effect as a GET when received, parsed, and executed by the device.

Command Syntax: ✷TRG

Type of Command: Aborts all previous measurement commands if not ✷WAI is used, and if not the factory default setting is changed with ✷DDT.

Complies to standards: IEEE 488.2 1987.

How to use the Trigger System

The SCPI TRIGger subsystem enables synchronization of instrument actions with specified internal or external events. The following list gives some examples.

Instrument action examples of events to synchronize with:

- measurement
- bus trigger
- external signal level or pulse
- 10 occurrences of a pulse on the external trigger input
- other instrument ready
- signal switching
- input signal present
- 1 second after input signal is present
- sourcing output signal
- switching system ready

The ARM-TRIG trigger configuration

Figure 18-1 gives a typical trigger configuration, the ARM-TRIG model. The configuration contains two event detection layers: the 'Wait for ARM' and 'Wait for TRIG' states.

This trigger configuration is sufficient for most instruments. More complex instruments, such as the PM 6680, have more ARM layers. The 'Wait for TRIG' event detection layer is always the last to be crossed before instrument actions can take place.

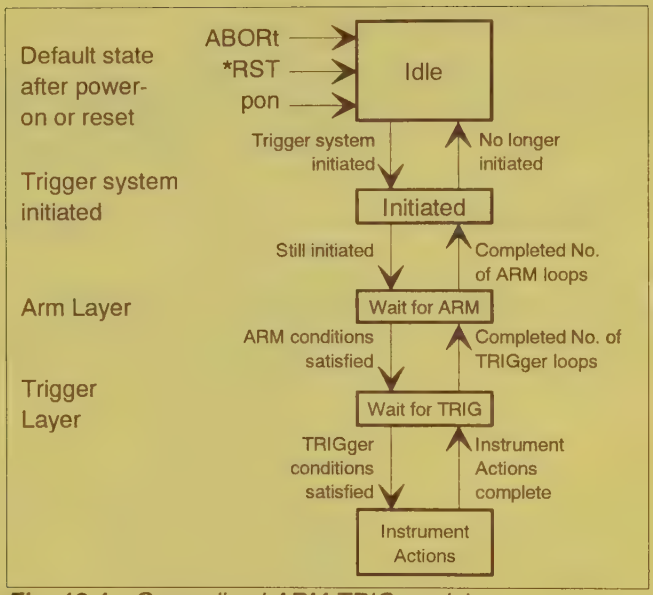


Fig. 18-1 *Generalized ARM-TRIG model.*

Structure of the IDLE and INITIATED States

When you switch on the power or send ✱RST or ABORT to the instrument, it sets the trigger system in the IDLE state; see figure 18-2.

The trigger system will exit from the IDLE state when the instrument receives an INITiate:IMMediate. The instrument will pass directly through the INITIATED state downward to the next event detection layers (if the instrument contains any more layers).

The trigger system will return to the INITIATED state when all events required by the detection layers have occurred and the instrument has made the intended measurement. When you program the trigger system to INITiate:CONTinuous ON, the instrument will directly exit the INITIATED state

moving downward and will repeat the whole flow described above. When INITiate:CONTinuous is OFF, the trigger system will return to the IDLE state.

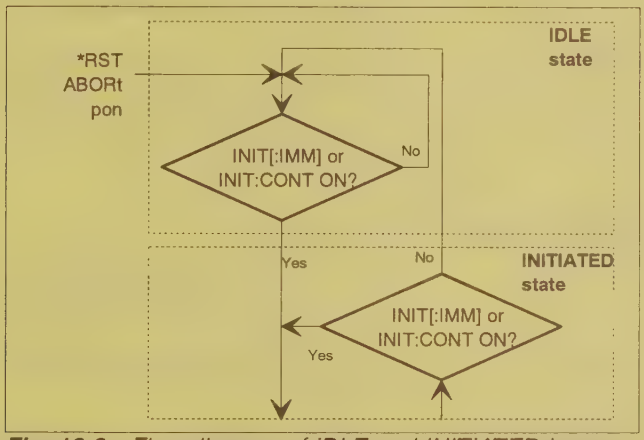


Fig. 18-2 *Flow diagram of IDLE and INITIATED layers.*

- *Structure of an Event Detection Layer*

The general structure of all event detection layers is identical and is roughly depicted by the flow diagram in figure 18-3

In each layer there are several programmable conditions, which must be satisfied to pass by the layer in a downward direction:

- *Forward Traversing an Event Detection Layer*

After initiating some loop counters, the instrument waits for the event to be detected. You can select the event to be detected by using the <layer>:SOURce command. For example :ARM:LAYer2:SOURce BUS

You can specify a more precise characteristic of the event to occur. For example: :ARM:LAYer:DELay 0.1

The instrument waits for a next event again if a multiple number of events must occur before the layer can be passed. This can be programmed by using the <layer>:ECOunt command.

You may program a certain delay between the occurrence of the event and entering into the next layer (or starting the device actions when in the TRIGger layer). This delay can be programmed by using the <layer>:DELay command.

- *Backward traversing an event detection layer*

The number of times a layer event has to initiate a device action can be programmed by using the <layer>:COUNt command. For example: :TRIGger:COUNt 3 causes the instrument to measure three times, each measurement being triggered by the specified events.

Possible trigger configurations

- *Simple*

An instrument may have the most simple trigger configuration without event detection layers. An INITiate command to the instrument will then directly start instrument actions.

- *Typical*

Many instruments will have the above depicted ARM-TRIG configuration.

- *Complex*

An instrument's trigger system may have several ARM layers, moreover the instrument may have several trigger sequences. Additional trigger sequences have the same layer structure. An application of an additional TRIGger SEQuence may be the distinct control of start and stop of a device action. In theory, the instrument may have an unlimited number of TRIGger sequences.

Triggering

- *✱TRG Trigger command*

The trigger command has the same function as the Group Execute Trigger command GET, defined by IEEE 488.1.

When to use ✱TRG and GET

The ✱TRG and the GET commands have the same effect on the instrument. If the Timer/Counter is in idle, i.e., not parsing or executing any commands, and the ✱DDT has not been used to redefine the trigger command, GET will execute much (≈ 20 μs) faster than ✱TRG (≈ 4 ms) since the instrument must always parse ✱TRG.

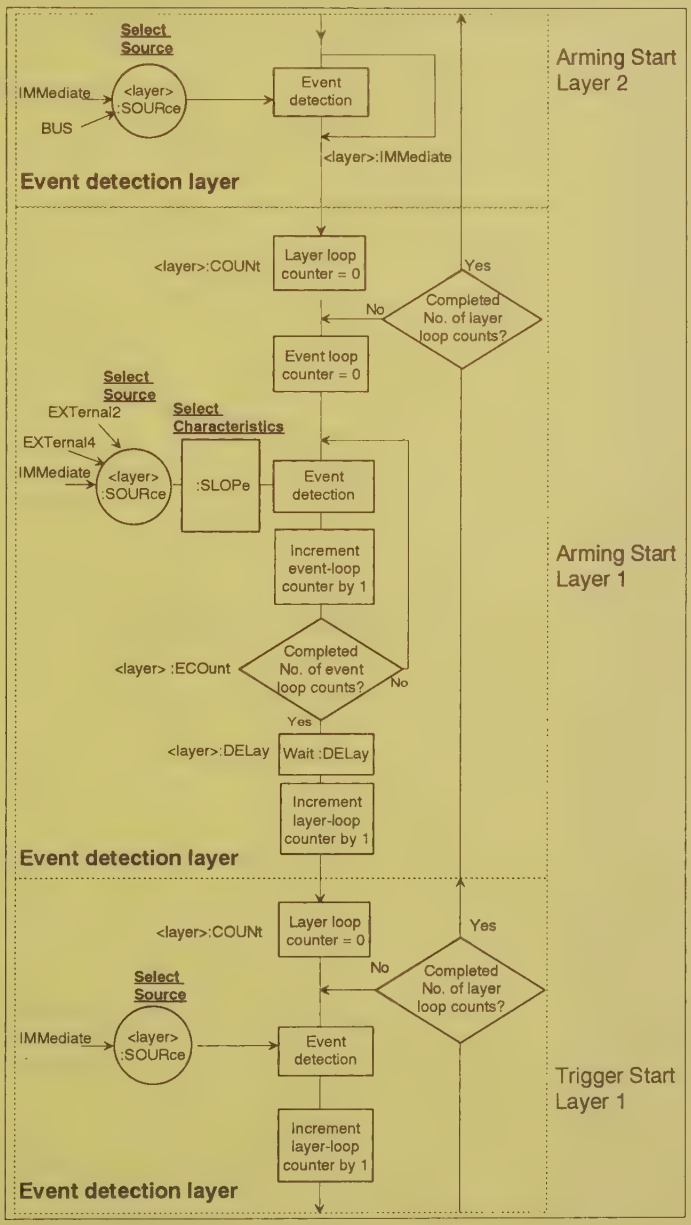


Fig. 18-3
Flow diagram of the PM 6680 Event Detection Layers.

This page is intentionally left blank.

Chapter 19

How to Measure

Introduction

The PM 6680 timer/counter can complete a measurement cycle in many different ways, each with its own advantage. This means that your first step is to select a basic "measurement scenario" based on the requirements of the measurements. This chapter contains some measurement scenarios that you can choose from.

PM 6680 can measure with an impressive speed if you program it correctly. You will find guidelines for speed improvements in each of the described measurement scenarios.

Controller synchronization

The start of measurements can either be individually or block synchronized by the controller. The instrument-to-controller synchronization deals with how to start a measurement or sequence of measurements and to read data in the most efficient way. You can also synchronize the measurement with the measuring object more accurately by using external control (arming), but this is not described here.

Some basic PM 6680 Commands

This chapter describes some basic PM 6680 commands that control the measurement sequence. It also roughly describes the trigger subsystem. The trigger subsystem is the functional part of the PM 6680 that controls the start and stop of measurements. This is the function that the controller interacts with when it controls the measurement sequence.

A simplified model of the PM 6680's trigger subsystem is a state-machine with four different states. These states are:

State	Purpose
IDLE	The counter waits for new commands. It is not measuring
WAIT_FOR_ BUS_ ARM	The counter is ready to receive a bus arming signal, GET or ✷TRG.
WAIT_ FOR_ MEASUREMENT_TO_ START	The counter waits for the input signal triggering to start the measurement or block of measurements. If the counter uses arming, it is waiting for the specified arming event.
MEASUREMENT	The counter measures. It monitors the hardware and controls the measurement time. If block measurement mode is used, (ARM:COUN or TRIG:COUN ≥2) the counter stays in this state until all measurements inside the block has been made.

Different actions cause the trigger subsystem to change between the different states. The transitions are shown in figure 19-4 . The status is reflected in status byte : STATus:OPERation:CONDition.

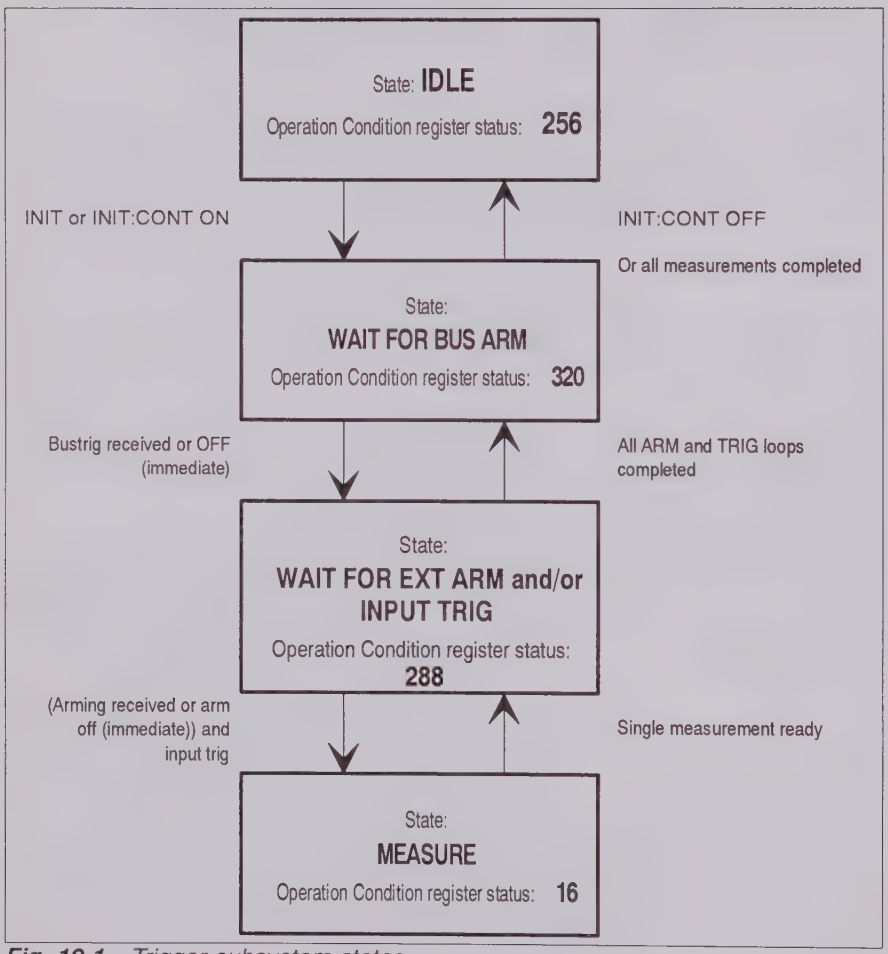


Fig. 19-1 *Trigger subsystem states*

CONFigure

The CONFigure command sets up the counter to do the measurement specified by the parameters of the command. The command gives a limited number of parameter options such as:

- Measurement function
- Measurement channel
- Number of measurements and sometimes also the:
  - Measuring time
  - Trigger level

The counter sets up the rest of its functions in the best way for the requested measurement. This means that any instrument setting may be changed by this command.

Examples:

- Set up to measure frequency:
  ```
  CONF:FREQ
  ```
- Set up to do 100 frequency measurements:
  ```
  CONF:ARRAY:FREQ␣(100)
  ```
- Set up to do 100 frequency measurements on the B-channel:
  ```
  CONF:ARRAY:FREQ␣(100),(@2)
  ```
- Set up to do 100 frequency measurements on the B-channel. Expected frequency 10 MHz that should be measured with a resolution of 1 Hz.
  ```
  CONF:ARRAY:FREQ␣(100),10e6,1,(@2)
  ```

INITiate

The INITiate command will normally start a measurement or measurement sequence and store the result internally in the PM 6680. However the actual action is to change the state of the trigger subsystem from "idle" to "wait_for_bus_arming". The result of changing the state of the trigger subsystem depends on the programming of this subsystem; For example it could be programmed to:

- Make 1000 measurements
- Wait for a GET/*TRG and then start a measurement
- Wait for a GET and then make 534 measurements

- Wait for an arming pulse and make one measurement
- Wait for an arming pulse and make 234 measurements

INITiate :CONTinuous

This command sets the counter in a mode where it continues with a new measurement immediately after it has finished the previous one. This is done by the trigger subsystem not returning to the "idle" state.

ABORt

This command stops the current measurement (if any), and sets the trigger subsystem to the "idle" state. This means that the counter is only waiting for new commands.

FETCh?

The FETCh query retrieves measurement data. It could either be a single value (SCALar) or a series of values (ARRay).

Examples:

- Get one measurement:

```
FETCh?
```

- Get 100 measurements:

```
FETCH:ARRAY? 100
```

NOTE: The number of measurements is defined by the setting of the ARM and TRIG counters. These can be set directly by the :CONF and :MEAS commands. The :FETCh:ARRay? query parameter only decides how many measurement results to read out.

READ?

This command simply means to start a measurement or measurement sequence and read data.

This is a compounded query identical to:

```
ABORt;INITiate;FETCh
```

This means that the counter starts a measurement (single or array) after it has aborted any previous measurements. It also returns the result.

Examples:

- Start one measurement and fetch result:
  `READ?`
- Start measurements and fetch 100 results:
  `READ:ARRAY? 100`

MEASure?

This is a compounded query consists of:

```
CONFigure;READ?
```

This means that the command sets up the counter and starts a measurement/measurement sequence.

Examples:

- Make a frequency measurement:
  `MEAS:FREQ?`
- Make 100 frequency measurements:
  `MEAS:ARRAY:FREQ? (100)`
- Make 100 frequency measurements on the B-channel:
  `MEAS:ARRAY:FREQ? (100),(@2)`
- Make 100 frequency measurements on the B-channel. Expected frequency 10 MHz that should be measured with a resolution of 1 Hz.
  `MEAS:ARRAY:FREQ? (100),10e6,1,(@2)`

Basic Measurement Method

A basic measurement method for a system composed of signal sources, measuring object, and measuring devices will be a simple step-by-step procedure. This procedure goes as follows:

Step 1: Set up signal sources

Step 2: Set up measurement devices

Step 3: Trigger measurement devices

Step 4: Read data

Step 5: Evaluate data

The above procedure may be repeated as many times as required.

The methods described here deal with how you should do steps 3 and 4 in the best and most efficient way with the PM 6680.

Individually Synchronized Measurements

This is a method that you should use when you need to start each measurement externally from the controller. The most probable reason that you should use individually synchronized measurements is that you need to evaluate data in real time and take decisions depending on the acquired data. An example of this could be to tune an oscillator by measuring the output frequency, and adjust the oscillator depending on the measured frequency.

Of the many available ways to do this with the PM 6680, three should be mentioned:

- *READ?*

The READ? query provides a basic mechanism for this. It ensures that the measurement is started after the counter receives the command. It will also send back the result. The READ? query should be preceded by setting up the counter

by using either CONFigure or individual programming commands. This command should be used if *no special speed requirements exists.*

- *INIT:CONT and GET*

In this method the trigger function is continuously initiated by the command INITiate:CONTinuous. This gives you the minimal firmware overhead if you don't change settings in the counter. Set up the counter either by using CONFigure, or by using individual programming commands before starting the measuring sequence. Setting up includes switching on the "wait for bus trigger" function with the command: ARM:START:LAY2:SOURCE _BUS.

As default, the counter starts a measurement and sends the result to the controller when receiving a GET or a *TRG command. This can be changed if required; however, the default action is very well fitted for this method. This method is the *fastest way to make individually synchronized measurements.*

- *MEASure?*

The MEASure query ensures that the measurement is started after the command is received. It also sends the result to the controller. It also includes setting up the counter. This command has the highest possible degree of compatibility to other instruments; however the command reprograms the counter, and often you need to set up the counter by yourself. This is primarily why we recommend the READ? query.

Block Synchronized Measurements

In the block synchronized mode, the controller only starts a sequence of measurements. The counter then measures, without any controller intervention, at the highest possible speed. It "dumps" the results into internal memory and reads them out for evaluation later. This method gives the highest possible data capture speed.

- *READ:ARRAY?*

This is the basic method for starting up a measurement sequence and reading the data. Set up the counter using CONFigure or individual programming commands before sending the READ:ARRAY? query. This method will make the measurements with a high measurement rate. The

speed depends on a number of individual measurement parameters, see also "General speed improvements" below. The counter stores the data in its internal memory and when it has captured all data, then it transfers the resulting array to the controller.

- *INIT + GET + FETCH:ARRAY?*

The READ:ARRAY? method has one drawback, it includes some unwanted firmware overhead between when the counter receives the command and it starts the first measurement. This can be solved by setting up the counter to wait for a GET before it starts the measurement sequence. The default actions for GET include sending a single result (the first value) when the counter has completed the sequence. This makes it possible to let the controller wait for the Message AVailable status bit to find out when the data capture is ready. You can read the complete array by using the FETCH:ARRAY command.

- *MEAS:ARRAY?*

The MEASure ARRay query ensures that the measurement sequence is started after the command is received. It will also send back the results. It also includes setting up the counter. This is the command that has the highest possible degree of compatibility with other instruments; however, this command reprograms the counter and often you need to program the counter yourself. This is why we recommend using the
READ:ARRAY? query.

General Speed Improvements

The PM 6680 has many options to improve measurement speed. Here you will get a list of actions that you can use to improve the measurement speed. Most of these commands *decrease the average dead time*. The dead time is the time between measurements, that is, from stop to the next start. These actions are all general, that is, they affect the rate of measurements for all measurement methods given above; however, they are especially valuable for the block synchronized measurements. In this mode, the dead time can be as low as 500 µs.

- *Display control*

The display can be switched off with the command DISPlay:ENABle_OFF. When you switch off the display, the

counter loses the measurement data resolution information. This means that the counter always sends all digits independently of whether or not they are significant. This command reduces the dead time by about 7 ms.

- *Real time calculation*

Normally the counter calculates the results in "real time." This means that for each measurement, the counter immediately calculates the result based on the raw data information in various counting registers. It needs to do this in order to display the result, make mathematical calculations, limit testing and statistical calculations. It is possible to defer the calculations until the controller requests these values. The counter intermediately stores the measurement data in a packed format. This is done with the command: :SENSe:INTernal:FORMat_PACKed.

This is the most important command when you want to improve the measurement rate.

- *GPIB data format*

You can select the format of the result sent on the GPIB. Two options exist: ASCii and REAL. The REAL format saves a lot of time both in the instrument and the controller for converting data. However, when the counter uses the REAL format, you lose the measurement data resolution information. It sends the REAL format as a block data element. This means that it sends data as: #18< 8 bytes real>.

The <8 bytes real> is a double precision binary floating point code according to IEEE 488.2 / IEEE 754.

- *Time measurement resolution*

The basic time measurement resolution can be selected by using the command SENSe:ACQ:RES_HIGH/LOW. When you set the resolution to low it will be 100 ns instead of the high resolution, which is 0.5 ns. If the counter does real-time calculations and you switch to 100 ns resolution, the measurement dead time decreases about 0.6 to 0.9 ms. If the counter does not do real-time calculations, then it saves only about 0.05 ms per measurement cycle. If the counter does real time calculations with the display switched on, then you can save up to 2 ms by selecting the lower resolution.

- *Automatic interpolator calibration*

The time interpolation technique achieves the high time resolution. The counter automatically calibrates these interpolators once per measurement cycle. You can control this automatic calibration by using the command CALibration:INTerpolator:AUTO ON/OFF. When you switch calibration off, the measurement cycle decreases.

Disabling this calibration makes the counter more sensitive to temperature changes. The measurement values may drift away, which results in a larger inaccuracy. However, when the instrument has been switched on for more than 20 minutes and the ambient temperature is stable within ± 5°C, this is no problem. You can also easily recalibrate the interpolators by using the CAL:INT:AUTO ONCE command.

- *Block measurements*

When dumping measurement results into the internal memory, it is important to program the arming and triggering counters in the best way. For maximum measurement rate use the block armed mode. To do this set:
:ARM:STARt:LAYer:COUNt_1 and TRIG:COUNt_N
where N is the number of measurements in a block (up to 2500).

Speed summary

The following table gives a summary of the dead time between measurements for some situations.

Speed Improvement Actions	Dead Time between Measurements		
	#1 Individually synchronized, INIT:CONT and GET including transfer to controller		*#2 Block synchronized*
	REAL	ASCII	REAL/ASCII
None	45 ms (READ?)	50 ms (READ?)	9 ms
Display off, 100 ns resolution, CAL:INT:AUTO OFF , Real time calculation off, block armed mode	8 ms	12 ms	0.5 ms
Display off, 0.5 ns resolution, CAL:INT:AUTO OFF, Real time calculation off, block armed mode	9 ms	12 ms	0.5 ms
Display off, 0.5 ns resolution, CAL:INT:AUTO OFF, Real time calculation on, block armed mode	9 ms	12 ms	20 ms
Display on, 0.5 ns resolution, CAL:INT:AUTO ON, Real time calculation on	15 ms	19 ms	9 ms

NOTE 1: The GPIB format command will not affect the dead time for the block synchronized mode because the counter captures all data before it transfers it to the controller.

NOTE 2: Switching the real time calculations on/off will not affect the dead time in mode #1. This is true because the calculations are still done inside the measurement loop during the output of data. In the block synchronized mode this command significantly decreases the dead time; however, the time for calculations (2 ms) is added to the transfer time.

This page is intentionally left blank.

Chapter 20

Programming Examples

Introduction

Each program example in this chapter is written for IBM-PC compatible computers equipped with either the Philips PM 2201 interface or the National Instruments PC-IIA. In addition to that many of the examples are written in both 'GW-BASIC' and 'C'.

Even if you do not have one of these interface boards or uses these computer languages, look at the examples anyway, they give you a good insight in how to program the instrument efficiently.

NOTE: To be able to run these programs without modification, the address switches on the rear panel of your timer/counter must be set to 10.

Example 1, Statistics

Example 2, Limit testing

Example 3, REAL data format

Example 4, Frequency profiling

Example 5, Fast sampling

Example 6, Status Reporting

GW-Basic examples for PM 2201

Setting up the interface

All these programs start with a declaration containing 55 lines of setup information for the interface. This declaration must be merged with the program prior to running them. The declaration is printed below, but it is also available as a file on the diskettes delivered with your interface. The file name is DECL.BAS.

```
10  'Copyright 1986, N.V. Philips Gloeilampenfabrieken
20  '
30  CLEAR ,64000!
40  INIT1 = 64000! : INIT2 = INIT1 + 2
50  DIM A%(26)
60  BLOAD "IOBIB.M",INIT1
70  CALL INIT1(A%(0),A%(1),A%(2),A%(3),A%(4),A%(5),A%(6),A%(7),A%(8),A%(9)
,A%(10),A%(11),A%(12),A%(13),A%(14),A%(15),A%(16),A%(17),A%(18),A%(19),A%
(20),A%(21),A%(22),A%(23),A%(24))
80  CALL INIT2(A%(25),A%(26))
90  '
100 IOINIT       = INIT1 + A%(0)
110 IORESET      = INIT1 + A%(1)
120 IOABORT      = INIT1 + A%(2)
130 IOCONTROL    = INIT1 + A%(3)
140 IOEOI        = INIT1 + A%(4)
150 IOEOL        = INIT1 + A%(5)
160 IOGETTERM    = INIT1 + A%(6)
170 IOMATCH      = INIT1 + A%(7)
180 IOOUTPUTS    = INIT1 + A%(8)
190 IOOUTPUT     = INIT1 + A%(9)
200 IOOUTPUTA    = INIT1 + A%(10)
210 IOENTERS     = INIT1 + A%(11)
220 IOENTER      = INIT1 + A%(12)
230 IOENTERA     = INIT1 + A%(13)
240 IOSEND       = INIT1 + A%(14)
250 IOSPOLL      = INIT1 + A%(15)
260 IOSTATUS     = INIT1 + A%(16)
270 IOTIMEOUT    = INIT1 + A%(17)
280 IOREMOTE     = INIT1 + A%(18)
290 IOLOCAL      = INIT1 + A%(19)
300 IOLLOCKOUT   = INIT1 + A%(20)
```

```
310 IOCLEAR      = INIT1 + A%(21)
320 IOTRIGGER    = INIT1 + A%(22)
330 IOGTS        = INIT1 + A%(23)
340 IORSV        = INIT1 + A%(24)
350 IOWAIT       = INIT1 + A%(25)
360 '
370 DEF.ERR      = INIT1 + A%(26)
380 PCIB.ERR$    = SPACE$(40)
390 CALL DEF.ERR(PCIB.ERR, PCIB.ERR$)
400 FALSE        = 0
410 TRUE         = 1
420 NOERR        = 0
430 EOFLW        = 14
440 EUNKNOWN     = 100001!
450 ESEL         = 100002!
460 ERANGE       = 100003!
470 ETIME        = 100004!
480 ECTRL        = 100005!
490 EPASS        = 100006!
500 ENUM         = 100007!
510 EADDR        = 100008!
520 '
530 ERASE A%
540 '
550 'Start application program after this line
```

Example 1, Statistics

In this example, the counter makes 10000 measurements and uses the statistical functions to determine MAX, MIN, MEAN, and Standard Deviation. All four results are sent to the controller.

```
560 '
570 AD = 7
580 CNTADR = 710
590 SC = 1
600 CALL IOINIT(AD, SC)
610 '
620 '
630 WRT$ = "*RST; *CLS; *SRE 16; :FUNC 'Freq 1'; :ACQ:APER 800E-9"
640 N = LEN(WRT$)
650 CALL IOOUTPUTS (CNTADR, WRT$, N)
660 WRT$ = ":INP:LEV:AUTO Off"
670 N = LEN(WRT$)
680 CALL IOOUTPUTS (CNTADR, WRT$, N)
690 '
700 ' ---- Enable statistics on 10000 measurements ----
710 WRT$ = ":CALC:AVER:STAT ON; COUN 10000"
720 N = LEN(WRT$)
730 CALL IOOUTPUTS (CNTADR, WRT$, N)
740 '
750 ' ==== Start  measurement ====
760 WRT$ = ":Init; *OPC?"
770 N = LEN(WRT$)
780 CALL IOOUTPUTS (CNTADR, WRT$, N)
790 '
800 ' ---- Wait for operation complete (MAV) ----
810 PRINT "WAITING FOR MEASUREMENT TO GET READY"
820 EVENT = 1
830 CALL IOWAIT (AD, EVENT, SPR)
840 '
850 ' ---- Read status and response ----
860 CALL IOSPOLL (CNTADR, SPR)
870 '
880 ' ----  Maximum ----
890 WRT$ = ":CALC:AVER:TYPE MAX; :CALC:IMM?"
900 N = LEN(WRT$)
910 CALL IOOUTPUTS (CNTADR, WRT$, N)
```

```
920 CALL IOENTERS (CNTADR, RESULT)
930 PRINT "MAXIMUM = "; RESULT
940 '
950 ' ----  Minimum ----
960 WRT$ = ":CALC:AVER:TYPE MIN; :CALC:IMM?"
970 N = LEN(WRT$)
980 CALL IOOUTPUTS (CNTADR, WRT$, N)
990 CALL IOENTERS (CNTADR, RESULT)
1000 PRINT "MINIMUM ="; RESULT
1010 '
1020 ' ----  Mean ----
1030 WRT$ = ":CALC:AVER:TYPE MEAN; :CALC:IMM?"
1040 N = LEN(WRT$)
1050 CALL IOOUTPUTS (CNTADR, WRT$, N)
1060 CALL IOENTERS (CNTADR, RESULT)
1070 PRINT "MEAN ="; RESULT
1080 '
1090 ' ----  Standard deviation ----
1100 WRT$ = ":CALC:AVER:TYPE SDEV; :CALC:IMM?"
1110 N = LEN(WRT$)
1120 CALL IOOUTPUTS (CNTADR, WRT$, N)
1130 CALL IOENTERS (CNTADR, RESULT)
1140 PRINT "STANDARD DEVIATION ="; RESULT
1150 END
1160 O CALL IOENTER (CNTADR, RESULT)
```

Example 2, Limit testing

This program uses limit testing to check that the frequency is above a preset value.

```
560 '
570 AD = 7
580 CNTADR = 710
590 SC = 1
600 CALL IOINIT(AD, SC)
610 '
620 '
630 ' ---- Set continuous frequency measurement ----
640 WRT$ = "*RST; *CLS; :FUNC 'FREQ 1'; :INIT:CONT ON"
650 N = LEN(WRT$)
660 CALL IOOUTPUTS (CNTADR, WRT$, N)
670 '
680 ' ---- Enable limit monitoring, limit 1 MHz ----
690 WRT$ = ":CALC:LIM ON; LIM:UPP 1E6; UPP:STATE ON"
700 N = LEN(WRT$)
710 CALL IOOUTPUTS (CNTADR, WRT$, N)
720 WRT$ = ":STAT:DREGO:ENAB 2; *SRE 1"
730 N = LEN(WRT$)
740 CALL IOOUTPUTS (CNTADR, WRT$, N)
750 '
760 ' ---- Wait until the limit is passed ----
770 PRINT "Waiting for limit to be passed"
780 EVENT = 1
790 SPR = 0
800 CALL IOWAIT (AD, EVENT, SPR)
810 '
820 ' ---- Read status and device status register ----
830 CALL IOSPOLL (CNTADR, SPR)
840 WRT$ = ":STAT:DREGO:EVEN?" +NL$
850 N = LEN(WRT$)
860 CALL IOOUTPUTS (CNTADR, WRT$, N)
870 CALL IOENTER (CNTADR, RESULT)
880 '
890 ' ---- Read frequency ----
900 WRT$ = "READ?"
910 N = LEN(WRT$)
```

```
920 CALL IOOUTPUTS (CNTADR, WRT$, N)
930 CALL IOENTER (CNTADR, RESULT)
940 PRINT "Frequency = "; RESULT
950 '
960 ' ---- Disable continuous measurement ----
970 WRT$ = ":INIT:CONT OFF"
980 N = LEN(WRT$)
990 CALL IOOUTPUTS (CNTADR, WRT$, N)
1000 END
```

Example 4, Frequency profiling

Frequency profiling visualizes frequency variations for a certain time. This program gives an output file called: PROFILE.DAT. If this file is imported to a spreadsheet program, for instance Lotus 123, you can create a graph like in the figure below.

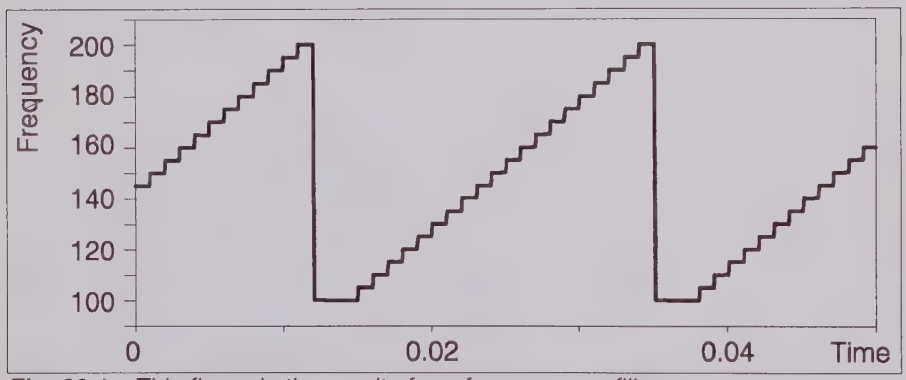


Fig. 20-1 *This figure is the results from frequency profiling on a sweep generator.*

```
560 '
570 AD = 7
580 CNTADR = 710
590 SC = 1
600 CALL IOINIT(AD, SC)
610 '
620 '
630 OPEN "O", 1, "PROFILE.DAT"
640 '
650 '
660 ' ---- Enable arming, etc. ----
670 WRT$ = ":TRIG:COUN 1; :ARM:COUN 1; SOUR EXT2"
680 N = LEN(WRT$)
690 CALL IOOUTPUTS(CNTADR, WRT$, N)
700 WRT$ = ":INP:LEV:AUTO ONCE; :INP2:LEV:AUTO ONCE"
710 N = LEN(WRT$)
```

```
720 CALL IOOUTPUTS(CNTADR, WRT$, N)
730 WRT$ = ":DISP:ENAB OFF; :ACQ:APER 1E-6"
740 N = LEN(WRT$)
750 CALL IOOUTPUTS(CNTADR, WRT$, N)
760 '
770 ARMDELAY = .0000002
780 '
790 ' ==== CAPTURE PROFILE =====
800 '
810 PRINT "Profiling"
820 '
830 FOR I=0 TO 999
840     ' ---- Set arming delay time ----
850     WRT$ = ":ARM:DEL" + STR$(ARMDELAY)
860     N = LEN(WRT$)
870     CALL IOOUTPUTS(CNTADR, WRT$, N)
880     '
890     ' ---- Measure and read result ----
900     WRT$ = "READ?"
910     N = LEN(WRT$)
920     CALL IOOUTPUTS(CNTADR, WRT$, N)
930     CALL IOENTER(CNTADR, RESULT)
940     '
950     ' ---- Write arming delay time and result to file ----
960     PRINT#1, STR$(ARMDELAY), RESULT
970     '
980     ' ---- Increase arming delay ----
990     ARMDELAY = ARMDELAY + .0000001
1000 NEXT I
1010 '
1020 WRT$ = ":DISP:ENAB ON"
1030 N = LEN(WRT$)
1040 CALL IOOUTPUTS(CNTADR, WRT$, N)
1050 '
1060 CLOSE 1
1070 END
```

Example 5, Fast sampling

This program makes a quick array measurement and stores the results in the internal memory of the counter. Then it writes the results to a file called MEAS.DAT. The measurement results as a function of the samples can be visualized in a spreadsheet program like Lotus 123 etc.

```
560 '
570 AD = 7
580 CNTADR = 710
590 SC = 1
600 CALL IOINIT(AD, SC)
610 '
620 '
630 OPEN "O", 1, "MEAS.DAT"
640 '
650 '
660 ' ---- Clear status ----
670 WRT$ = "*CLS"
680 N = LEN(WRT$)
690 CALL IOOUTPUTS (CNTADR, WRT$, N)
700 '
710 ' ---- Enable 1000 measurement with maximum speed ----
720 WRT$ = ":TRIG:COUN 1000; :ARM:COUN 1"
730 N = LEN(WRT$)
740 CALL IOOUTPUTS (CNTADR, WRT$, N)
750 WRT$ = ":INP:LEV:AUTO ONCE; :CAL:INT:AUTO OFF"
760 N = LEN(WRT$)
770 CALL IOOUTPUTS (CNTADR, WRT$, N)
780 WRT$ = ":DISP:ENAB OFF; :INT:FORM PACKED"
790 N = LEN(WRT$)
800 CALL IOOUTPUTS (CNTADR, WRT$, N)
810 WRT$ = ":ACQ:APER 800E-9; :AVER:STAT OFF"
820 N = LEN(WRT$)
830 CALL IOOUTPUTS (CNTADR, WRT$, N)
840 '
850 ' ---- Enable SRQ on operation complete ----
860 WRT$ = "*ESE 1; *SRE 32"
870 N = LEN(WRT$)
880 CALL IOOUTPUTS (CNTADR, WRT$, N)
```

```
890 '
900 ' ---- Start measurement ----
910 PRINT "Measuring"
920 WRT$ = "INIT; *OPC"
930 N = LEN(WRT$)
940 CALL IOOUTPUTS (CNTADR, WRT$, N)
950 '
960 ' ---- Wait for operation complete ----
970 EVENT = 1
980 CALL IOWAIT (AD, EVENT, SPR)
990 '
1000 ' ---- Read status and event status register ----
1010 CALL IOSPOLL (CNTADR, SPR)
1020 WRT$ = "*ESR?"
1030 N = LEN(WRT$)
1040 CALL IOOUTPUTS (CNTADR, WRT$, N)
1050 CALL IOENTER (CNTADR, RESULT)
1060 '
1070 PRINT "Fetching result"
1080 '
1090 FOR I=0 TO 999
1100   ' ---- Fetch one result ----
1110   WRT$ = "FETCH?"
1120   N = LEN(  WRT$)
1130   CALL IOOUTPUTS (CNTADR, WRT$, N)
1140   CALL IOENTER (CNTADR, RESULT)
1150   '
1160   ' ---- Write result to file ----
1170   PRINT#1, I, RESULT
1180 NEXT I
1190 '
1200 WRT$ = ":DISP:ENAB ON"
1210 N = LEN(WRT$)
1220 CALL IOOUTPUTS(CNTADR, WRT$, N)
1230 '
1240 CLOSE 1
1250 END
```

Example 6, Status Reporting

This program sets up the status reporting for Service Request on 'Message Available' and 'Command', 'Execution' or Query' errors.

It reads a command from the controller keyboard and sends it to PM 6680, then it checks the status byte using Serial Poll. It determines the reason for Service Request, reads query responses and error messages.

```
560 '
570 AD = 7
580 CNTADR = 710
590 SC = 1
600 CALL IOINIT(AD, SC)
610 '
620 '
630 ' ---- CLEAR STATUS ----
640 WRT$ = "*cls"
650 N = LEN(WRT$)
660 CALL IOOUTPUTS (CNTADR, WRT$, N)
670 '
680 ' ---- SET EVENT STATUS ENABLE ----
690 ' Enable Command Error, Execution Error and Query Error
700 WRT$ = "*ese 52"
710 N = LEN(WRT$)
720 CALL IOOUTPUTS (CNTADR, WRT$, N)
730 '
740 ' ---- SET SERVICE REQUEST ENABLE ----
750 ' Enable Service Request on Event Status and Message Available
760 WRT$ = "*sre 48"
770 N = LEN(WRT$)
780 CALL IOOUTPUTS (CNTADR, WRT$, N)
790 '
800 ' ======== MAIN LOOP ========================================
810 WHILE 1
820     '
830     ' ---- ENTER COMMAND STRING AND SEND TO COUNTER ----
840     LINE INPUT "Enter command string (<CR> to end):", CMD$
850     IF CMD$ = "" GOTO 1370
860     CMD$ = CMD$
```

```
870    N = LEN(CMD$)
880    CALL IOOUTPUTS (CNTADR, CMD$, N)
890    ' WAIT for execution
900    FOR I=1 TO 1000
910       CALL IOSPOLL (CNTADR, SPR%)
920       IF SPR% AND 16 THEN GOTO 960
930    NEXT I
940    '
950    ' ---- READ STATUS BYTE ----
960    IF SPR% <> 0 THEN PRINT "Status byte = "; SPR%
970    ELSE GOTO 1360
980       '
990       ' ---- CHECK MESSAGE AVAILABLE BIT ----
1000      WHILE SPR% AND 16
1010         PRINT "  Message available bit set"
1020         MSG$ = SPACE$(255)
1030         CALL IOENTERS (CNTADR, MSG$, 255, N)
1040         LFPOS = INSTR(MSG$, CHR$(10))
1050         IF LFPOS <> 0 THEN PRINT "Response = " LEFT$(MSG$, LFPOS)
1060         IF LFPOS = 0 THEN PRINT "Response = "; MSG$
1070         CALL IOSPOLL (CNTADR, SPR%)
1080      WEND
1090      '
1100      ' ---- CHECK EVENT STATUS BIT ----
1110      IF NOT SPR% AND 32 GOTO 1360
1120         PRINT "  Event status bit set"
1130         WRT$ = "*esr?"
1140         N = LEN(WRT$)
1150         CALL IOOUTPUTS (CNTADR, WRT$, N)
1160         ESR$ = SPACE$(255)
1170         CALL IOSPOLL (CNTADR, SPR%)
1180         CALL IOENTERS (CNTADR, ESR$, 255, N)
1190         ESR% = VAL(ESR$)
1200         IF ESR% AND 32 THEN PRINT "    Command error"
1210         IF ESR% AND 16 THEN PRINT "    Execution error"
1220         IF ESR% AND 4  THEN PRINT "    Query error"
1230         '
1240         ' ---- READ ERROR MESSAGES ----
1250         WRT$ = "syst:err?"
1260         ERRMESS$ = SPACE$(255)
1270         N = LEN(WRT$)
1280         CALL IOOUTPUTS (CNTADR, WRT$, N)
1290         CALL IOENTERS (CNTADR, ERRMESS$, 255, N)
```

```
1300          WHILE NOT INSTR(ERRMESS$, "No error") <> 0
1310             PRINT LEFT$(ERRMESS$, INSTR(ERRMESS$, CHR$(10)))
1320             N = LEN(WRT$)
1330             CALL IOOUTPUTS (CNTADR, WRT$, N)
1340             CALL IOENTERS (CNTADR, ERRMESS$, 255, N)
1350          WEND
1360 WEND
1370 PRINT "PROGRAM TERMINATED"
1380 END
```

'C' examples for PM 2201

Example 1, Statistics

In this example, the counter makes 10000 measurements and uses the statistical functions to determine MAX, MIN, MEAN, and Standard Deviation. All four results are sent to the controller.

```
#include "cdecl.h"

#include <stdio.h>
#include <process.h>

main ()

{
  int                              CntAdr, Status, Length, i;
  char                             InString[80];

  CntAdr = 710;

  ioinit(7, 1);

  iooutputs(CntAdr, "*CLS; *SRE 16", 13);
  iooutputs(CntAdr, "*RST; :FUNC 'Freq 1'; :ACQ:APER 800E-9", 38);
  iooutputs(CntAdr, ":INP:LEV:AUTO OFF", 17);

  /*Enable statistics on 10000 measurements*/
  iooutputs(CntAdr, ":CALC:AVER:STAT ON; COUN 10000", 30);
  iooutputs(CntAdr, ":TRIG:COUN 10000", 16);

  /*Start measurement*/
  iooutputs(CntAdr, ":Init; *OPC?", 12);

  /*Wait for operation complete*/
  printf("Waiting for measurement to get ready\n");
  iowait(7, 1, &Status);

  /**Read status and response**/
  iospoll(CntAdr, &Status);
  Length = 80;
```

```
ioenters(CntAdr, InString, &Length);

/*Read maximum value**/
iooutputs(CntAdr, ":CALC:AVER:TYPE MAX; :CALC:IMM?", 31);
Length = 80;
ioenters(CntAdr, InString, &Length);
InString[Length] = '\0';
printf("Maximum = %s\n", InString);

/*Read minimum value*/
iooutputs(CntAdr, ":CALC:AVER:TYPE MIN; :CALC:IMM?", 31);
Length = 80;
ioenters(CntAdr, InString, &Length);
InString[Length] = '\0';
printf("Minimum = %s\n", InString);


/*Read mean value*/
iooutputs(CntAdr, ":CALC:AVER:TYPE MEAN; :CALC:IMM?", 32);
Length = 80;
ioenters(CntAdr, InString, &Length);
InString[Length] = '\0';
printf("Mean = %s\n", InString);

/*Read standard deviation value*/
iooutputs(CntAdr, ":CALC:AVER:TYPE SDEV; :CALC:IMM?", 32);
Length = 80;
ioenters(CntAdr, InString, &Length);
InString[Length] = '\0';
printf("Standard deviation = %s\n", InString);

exit(0);
}
```

Example 2, Limit testing

This program uses limit testing to check that the frequency is above a preset value.

```
#include "cdecl.h"

#include <stdio.h>
#include <process.h>

main ()

{
  int                              CntAdr, Status, Length, i;
  char                             InString[80];

  CntAdr = 710;

  ioinit(7, 1);

  /*Set continuous frequency measurement*/
  iooutputs(CntAdr, "*RST; *CLS; :FUNC 'Freq 1'; :INIT:CONT ON", 41);

  /*Enable limit monitoring, limit 1 MHz*/
  iooutputs(CntAdr, ":CALC:LIM ON; LIM:UPP 1E6; UPP:STAT ON", 38);
  iooutputs(CntAdr, ":STAT:DREG0:ENAB 2; *SRE 1", 26);

  /*Wait until the limit is passed*/
  printf("Waiting for limit to be passed\n");
  iowait(7, 1, &Status);

  /**Read status**/
  iospoll(CntAdr, &Status);
  iooutputs(CntAdr, ":STAT:DREG0:EVEN?", 17);
  Length = 80;
  ioenters(CntAdr, InString, &Length);

  /*Read frequency**/
  iooutputs(CntAdr, "READ?", 5);
  Length = 80;
  ioenters(CntAdr, InString, &Length);
```

```
  InString[Length] = '\0';
  printf("Frequency = %s\n", InString);

  /*Disable continuous measurement*/
  iooutputs(CntAdr, ":INIT:CONT OFF", 14);

  exit(0);
}
```

Example 3, REAL data format

This program uses the REAL data format to speed up the measurement.

IEEE 488.2 binary real format follows the 'little-indian' format with the most significant byte first and the least significant byte last. Intel processors use the 'big-indian' format, with the least significant byte first, so we have to reverse the byte order of the incoming block when running on a PC (Intel processor).

```
#include "cdecl.h"

#include <stdio.h>
#include <process.h>
#include <conio.h>

main ()

{
  int                              CntAdr, Length, i;
  char                             InString[80];
  double                           DoubleFreq;

  CntAdr = 710;

  ioinit(7, 1);

  /*Make the counter output it's result in real format*/
  iooutputs(CntAdr, ":FORM REAL", 10);

  /*Make continuous measurements until a key is hit*/
  do {

    /*Make a measurement and read the result*/
    iooutputs(CntAdr, "READ?", 5);
    Length = 80;
    ioenters(CntAdr, InString, &Length);
```

```
    /*Assign the bytes 3...10 of InString to DoubleFreq bytes 7...0.
    The format of InString is #18******** , where "********"
    represents the value.*/
    for (i=0; i<8; i++)
      ((unsigned char *)&DoubleFreq)[7-i] = InString[3+i];

    /*Print the result*/
    printf("%le\n", DoubleFreq);

  } while (!kbhit());

  /*Restore ascii output format*/
  iooutputs(CntAdr, ":FORM ASCII", 11);

  exit(0);
}
```

Example 4, Frequency profiling

Frequency profiling visualizes frequency variations for a certain time. This program gives an output file called: PROFILE.DAT. If this file is imported to a spreadsheet program, for instance Lotus 123, you can create a graph like in the figure below.

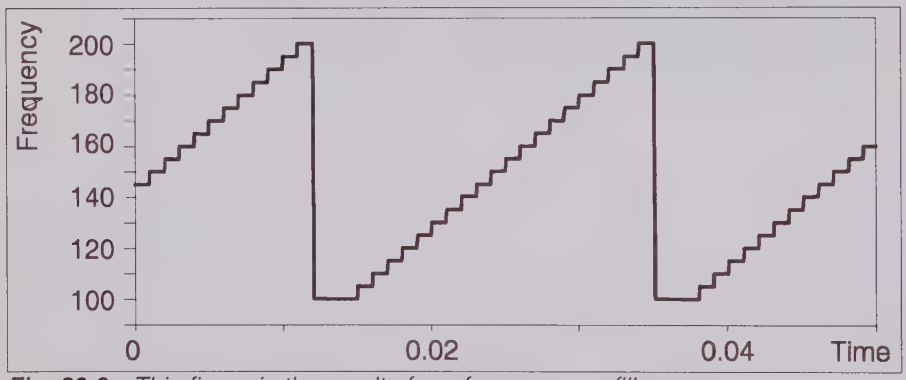


Fig. 20-2 *This figure is the results from frequency profiling on a sweep generator.*

```
#include "cdecl.h"

#include <stdio.h>
#include <process.h>
#include <string.h>

main ()

{
  int                          CntAdr, Length, i;
  char                         ArmString[80], InString[80];
  double                       ArmDelay;
  FILE                         *ofp;

  if (ofp = fopen("PROFILE.DAT", "w")) {
```

```
    CntAdr = 710;

    ioinit(7, 1);

    /*Enable arming, etc.*/
    iooutputs(CntAdr, ":TRIG:COUN 1; :ARM:COUN 1; SOUR EXT2", 36);
    iooutputs(CntAdr, ":INP:LEV:AUTO ONCE; :INP2:LEV:AUTO ONCE", 39);
    iooutputs(CntAdr, ":DISP:ENAB OFF; :ACQ:APER 1E-6", 30);

    ArmDelay=200e-9;

    /*CAPTURE PROFILE*/

    printf("Profiling");

    for (i=0; i<1000; i++) {
      /*Set arming delay time*/
      sprintf(ArmString, ":ARM:DEL %le", ArmDelay);
      iooutputs(CntAdr, ArmString, strlen(ArmString));

      /*Measure and read result*/
      iooutputs(CntAdr, "READ?", 5);
      Length = 80;
      ioenters(CntAdr, InString, &Length);
      InString[Length] = '\0';

      /*Write arming delay time and result to file*/
      fprintf(ofp, "%le, %s", ArmDelay, InString);

      /*Increase arming delay*/
      ArmDelay += 100e-9;
    }

    iooutputs(CntAdr, ":DISP:ENAB ON", 13);

    /*Close file*/
    fclose(ofp);

  } else
    printf("CANT OPEN FILE");

  exit(0);
}
```

Example 5, Fast sampling

This program makes a quick array measurement and stores the results in the internal memory of the counter. Then it writes the results to a file called MEAS.DAT. The measurement results as a function of the samples can be visualized in a spreadsheet program like Lotus 123 etc.

```
#include "cdecl.h"

#include <stdio.h>
#include <process.h>
#include <string.h>

main ()

{
  int                              CntAdr, Status, Length, i;
  char                             InString[80];
  FILE                             *ofp;

  if (ofp = fopen("MEAS.DAT", "w")) {

    CntAdr = 710;

    ioinit(7, 1);

    /*Clear status*/
    iooutputs(CntAdr, "*CLS", 4);

    /*Enable 1000 measurement with maximum speed*/
    iooutputs(CntAdr, ":TRIG:COUN 1000; :ARM:COUN 1", 28);
    iooutputs(CntAdr, ":INP:LEV:AUTO ONCE; :CAL:INT:AUTO OFF", 37);
    iooutputs(CntAdr, ":DISP:ENAB OFF; :INT:FORM PACKED", 32);
    iooutputs(CntAdr, ":ACQ:APER 800E-9; :AVER:STAT OFF", 31);

    /**Enable SRQ on operation complete**/
    iooutputs(CntAdr, "*ESE 1; *SRE 32", 15);

    /*Start measurement*/
    printf("Measuring\n");
```

```
    iooutputs(CntAdr, "INIT; *OPC", 11);

    /*Wait for operation complete*/
    iowait(7, 1, &Status);

    /**Read status and event status register**/
    iospoll(CntAdr, &Status);
    iooutputs(CntAdr, "*ESR?", 5);
    Length = 80;
    ioenters(CntAdr, InString, &Length);

    printf("Fetching result");

    for (i=0; i<1000; i++) {
      /*Fetch one result*/
      iooutputs(CntAdr, "FETCH?", 6);
      Length = 80;
      ioenters(CntAdr, InString, &Length);
      InString[Length] = '\0';

      /*Write result to file*/
      fprintf(ofp, "%d, %s", i, InString);
    }

    iooutputs(CntAdr, ":DISP:ENAB ON", 13);
    /*Close file*/
    fclose(ofp);

  } else
    printf("CANT OPEN FILE");

  exit(0);
}
```

This page is intentionally left blank.

GW-Basic examples for National Instruments PC-IIA

Setting up the interface

All these programs start with a declaration containing 3 lines of setup information for the interface. This declaration must be merged with the programs prior to running them. The declaration is printed below, but it is also available as a file on the diskettes delivered with your interface. The file name is DECL.BAS.

```
20 CLEAR ,60000! : IBINIT1=60000! : IBINIT2=IBINIT1+3 : BLOAD "bib.m",IBINIT1
30 CALL IBINIT1(IBFIND,IBTRG,IBCLR,IBPCT,IBSIC,IBLOC,IBPPC,IBBNA,IBONL,IBRSC,
IBSRE,IBRSV,IBPAD,IBSAD,IBIST,IBDMA,IBEOS,IBTMO,IBEOT,IBRDF,IBWRTF,IBTRAP,
IBDEV,IBLN)
40 CALL IBINIT2(IBGTS,IBCAC,IBWAIT,IBPOKE,IBWRT,IBWRTA,IBCMD,IBCMDA,IBRD,
IBRDA,IBSTOP,IBRPP,IBRSP,IBDIAG,IBXTRC,IBRDI,IBWRTI,IBRDIA,IBWRTIA,IBSTA%,
IBERR%,IBCNT%)
```

Example 1, Statistics

In this example, the counter makes 10000 measurements and uses the statistical functions to determine MAX, MIN, MEAN, and Standard Deviation. All four results are sent to the controller.

```
50 CNTNAME$ = "DEV10"
60 CALL IBFIND (CNTNAME$, CNT%)
70 '
80 '
90 WRT$ = "*RST; *CLS; *SRE 16; :FUNC 'Freq 1'; :ACQ:APER 800E-9"
100 CALL IBWRT (CNT%, WRT$)
110 WRT$ = ":INP:LEV:AUTO Off"
120 CALL IBWRT (CNT%, WRT$)
130 '
140 ' ---- Enable statistics on 10000 measurements ----
150 WRT$ = ":CALC:AVER:STAT ON; COUN 10000"
160 CALL IBWRT (CNT%, WRT$)
170 '
180 ' ==== Start  measurement ====
190 WRT$ = ":Init; *OPC?"
200 CALL IBWRT (CNT%, WRT$)
210 '
220 ' ---- Wait for operation complete (MAV) ----
230 PRINT "WAITING FOR MEASUREMENT TO GET READY"
240 MASK% = &H800
250 CALL IBWAIT (CNT%, MASK%)
260 '
270 ' ---- Read status and response ----
280 CALL IBRSP (CNT%, SPR%)
290 MSG$ = SPACE$(255)
300 CALL IBRD (CNT%, MSG$)
310 '
320 ' ----  Maximum ----
330 WRT$ = ":CALC:AVER:TYPE MAX; :CALC:IMM?"
340 CALL IBWRT (CNT%, WRT$)
350 MSG$ = SPACE$(255)
360 CALL IBRD (CNT%, MSG$)
370 PRINT "MAXIMUM = "; LEFT$(MSG$, IBCNT%)
380 '
```

```
390 ' ----  Minimum ----
400 WRT$ = ":CALC:AVER:TYPE MIN; :CALC:IMM?"
410 CALL IBWRT (CNT%, WRT$)
420 MSG$ = SPACE$(255)
430 CALL IBRD (CNT%, MSG$)
440 PRINT "MINIMUM ="; LEFT$(MSG$, IBCNT%)
450 '
460 ' ----  Mean ----
470 WRT$ = ":CALC:AVER:TYPE MEAN; :CALC:IMM?"
480 CALL IBWRT (CNT%, WRT$)
490 MSG$ = SPACE$(255)
500 CALL IBRD (CNT%, MSG$)
510 PRINT "MEAN ="; LEFT$(MSG$, IBCNT%)
520 '
530 ' ----  Standard deviation ----
540 WRT$ = ":CALC:AVER:TYPE SDEV; :CALC:IMM?"
550 CALL IBWRT (CNT%, WRT$)
560 MSG$ = SPACE$(255)
570 CALL IBRD (CNT%, MSG$)
580 PRINT "STANDARD DEVIATION ="; LEFT$(MSG$, IBCNT%)
590 END
```

Example 2, Limit testing

This program uses limit testing to check that the frequency is above a preset value.

```
50 CNTNAME$ = "DEV10"
60 CALL IBFIND (CNTNAME$, CNT%)
70 '
80 '
90 ' ---- Set continuous frequency measurement ----
100 WRT$ = "*RST; *CLS; :FUNC 'FREQ 1'; :INIT:CONT ON"
110 CALL IBWRT (CNT%, WRT$)
120 '
130 ' ---- Enable limit monitoring, limit 1 MHz ----
140 WRT$ = ":CALC:LIM ON; LIM:UPP 1E6; UPP:STATE ON"
150 CALL IBWRT (CNT%, WRT$)
160 WRT$ = ":STAT:DREGO:ENAB 2; *SRE 1"
170 CALL IBWRT (CNT%, WRT$)
180 '
190 ' ---- Wait until the limit is passed ----
200 PRINT "Waiting for limit to be passed"
210 MASK% = &H800
220 CALL IBWAIT (CNT%, MASK%)
230 '
240 ' ---- Read status and device status register ----
250 CALL IBRSP (CNT%, SPR%)
260 '
270 ' ---- Read frequency ----
280 WRT$ = "READ?"
290 CALL IBWRT (CNT%, WRT$)
300 MSG$ = SPACE$(255)
310 CALL IBRD (CNT%, MSG$)
320 PRINT "Frequency = "; LEFT$(MSG$, IBCNT%)
330 WRT$ = ":STAT:DREGO:EVEN?"
340 CALL IBWRT (CNT%, WRT$)
350 MSG$ = SPACE$(255)
360 CALL IBRD (CNT%, MSG$)
370 '
380 ' ---- Disable continuous measurement ----
390 WRT$ = ":INIT:CONT OFF"
400 CALL IBWRT (CNT%, WRT$)
410 END
```

Example 4, Frequency profiling

Frequency profiling visualizes frequency variations for a certain time. This program gives an output file called: PROFILE.DAT. If this file is imported to a spreadsheet program, for instance Lotus 123, you can create a graph like in the figure below.

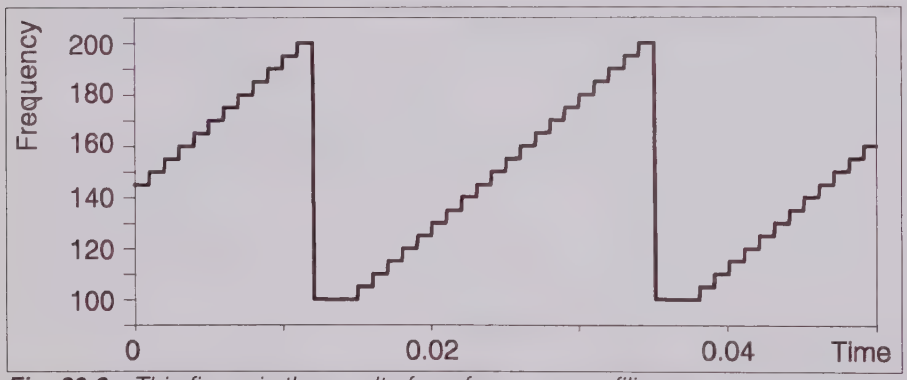


Fig. 20-3 *This figure is the results from frequency profiling on a sweep generator.*

```
50 '
60 OPEN "O", 1, "PROFILE.DAT"
70 CNTNAME$ = "DEV10"
80 CALL IBFIND (CNTNAME$, CNT%)
90 '
100 '
110 ' ---- Enable arming, etc. ----
120 WRT$ = ":TRIG:COUN 1; :ARM:COUN 1; SOUR EXT2"
130 CALL IBWRT(CNT%, WRT$)
140 WRT$ = ":INP:LEV:AUTO ONCE; :INP2:LEV:AUTO ONCE"
150 CALL IBWRT(CNT%, WRT$)
160 WRT$ = ":DISP:ENAB OFF; :ACQ:APER 1E-6"
170 CALL IBWRT(CNT%, WRT$)
180 '
190 ARMDELAY = .0000002
200 '
```

```
210 ' ==== CAPTURE PROFILE =====
220 '
230 PRINT "Profiling"
240 '
250 FOR I=0 TO 999
260     ' ---- Set arming delay time ----
270     WRT$ = ":ARM:DEL" + STR$(ARMDELAY)
280     CALL IBWRT(CNT%, WRT$)
290     '
300     ' ---- Measure and read result ----
310     WRT$ = "READ?"
320     CALL IBWRT(CNT%, WRT$)
330     MSG$ = SPACE$(255)
340     CALL IBRD(CNT%, MSG$)
350     '
360     ' ---- Write arming delay time and result to file ----
370     PRINT#1, STR$(ARMDELAY), LEFT$(MSG$, INSTR(MSG$, CHR$(10)))
380     '
390     ' ---- Increase arming delay ----
400     ARMDELAY = ARMDELAY + .0000001
410 NEXT I
420 '
430 WRT$ = ":DISP:ENAB ON"
440 CALL IBWRT(CNT%, WRT$)
450 '
460 CLOSE 1
470 END
```

Example 5, Fast sampling

This program makes a quick array measurement and stores the results in the internal memory of the counter. Then it writes the results to a file called MEAS.DAT. The measurement results as a function of the samples can be visualized in a spreadsheet program like Lotus 123 etc.

```
50 '
60 OPEN "O", 1, "MEAS.DAT"
70 CNTNAME$ = "DEV10"
80 CALL IBFIND (CNTNAME$, CNT%)
90 '
100 '
110 ' ---- Clear status ----
120 WRT$ = "*CLS"
130 CALL IBWRT(CNT%, WRT$)
140 '
150 ' ---- Enable 1000 measurement with maximum speed ----
160 WRT$ = ":TRIG:COUN 1000; :ARM:COUN 1"
170 CALL IBWRT (CNT%, WRT$)
180 WRT$ = ":INP:LEV:AUTO ONCE; :CAL:INT:AUTO OFF"
190 CALL IBWRT (CNT%, WRT$)
200 WRT$ = ":DISP:ENAB OFF; :INT:FORM PACKED"
210 CALL IBWRT (CNT%, WRT$)
220 WRT$ = ":ACQ:APER 800E-9; :AVER:STAT OFF"
230 CALL IBWRT (CNT%, WRT$)
240 '
250 ' ---- Enable SRQ on operation complete ----
260 WRT$ = "*ESE 1; *SRE 32"
270 CALL IBWRT (CNT%, WRT$)
280 '
290 ' ---- Start measurement ----
300 PRINT "Measuring"
310 WRT$ = "INIT; *OPC"
320 CALL IBWRT (CNT%, WRT$)
330 '
340 ' ---- Wait for operation complete ----
350 MASK = &H800
360 CALL IBWAIT (CNT%, MASK)
370 '
```

```
380 ' ---- Read status and event status register ----
390 CALL IBRSP (CNT%, SPR%)
400 WRT$ = "*ESR?"
410 CALL IBWRT (CNT%, WRT$)
420 MSG$ = SPACE$(255)
430 CALL IBRD (CNT%, MSG$)
440 '
450 PRINT "Fetching result"
460 '
470 FOR I=0 TO 999
480   ' ---- Fetch one result ----
490   WRT$ = "FETCH?"
500   CALL IBWRT (CNT%, WRT$)
510   MSG$ = SPACE$(255)
520   CALL IBRD (CNT%, MSG$)
530   '
540   ' ---- Write result to file ----
550   PRINT#1, I, LEFT$(MSG$, INSTR(MSG$, CHR$(10)))
560 NEXT I
570 '
580 WRT$ = ":DISP:ENAB ON"
590 CALL IBWRT(CNT%, WRT$)
600 '
610 CLOSE 1
620 END
```

Example 6, Status Reporting

This program sets up the status reporting for Service Request on 'Message Available' and 'Command', 'Execution' or Query' errors.

It reads a command from the controller keyboard and sends it to PM 6680, then it checks the status byte using Serial Poll. It determines the reason for Service Request, reads query responses and error messages.

```
50 CNTNAME$ = "DEV10"
60 CALL IBFIND (CNTNAME$, CNT%)
70 '
80 '
90 ' ---- CLEAR STATUS ----
100 WRT$ = "*cls"
110 CALL IBWRT (CNT%, WRT$)
120 '
130 ' ---- SET EVENT STATUS ENABLE ----
140 ' Enable Command Error, Execution Error and Query Error
150 WRT$ = "*ese 52"
160 CALL IBWRT (CNT%, WRT$)
170 '
180 ' ---- SET SERVICE REQUEST ENABLE ----
190 ' Enable Service Request on Event Status and Message Available
200 WRT$ = "*sre 48"
210 CALL IBWRT (CNT%, WRT$)
220 '
230 ' ======== MAIN LOOP ==========================================
240 WHILE 1
250    '
260    ' ---- ENTER COMMAND STRING AND SEND TO COUNTER ----
270    LINE INPUT "Enter command string (<CR> to end):", CMD$
280    IF CMD$ = "" GOTO 760
290    CMD$ = CMD$
300    CALL IBWRT (CNT%, CMD$)
310    ' WAIT for execution
320    FOR I=1 TO 1000
330       CALL IBRSP (CNT%, SPR%)
340       IF SPR% AND 16 THEN GOTO 380
350    NEXT I
```

```
360     '
370     ' ---- READ STATUS BYTE ----
380     IF SPR% <> 0 THEN PRINT "Status byte = "; SPR%
390     ELSE GOTO 750
400        '
410        ' ---- CHECK MESSAGE AVAILABLE BIT ----
420        WHILE SPR% AND 16
430           PRINT "  Message available bit set"
440           MSG$ = SPACE$(255)
450           CALL IBRD (CNT%, MSG$)
460           LFPOS = INSTR(MSG$, CHR$(10))
470           IF LFPOS <> 0 THEN PRINT "Response = " LEFT$(MSG$, LFPOS)
480           IF LFPOS = 0 THEN PRINT "Response = "; MSG$
490           CALL IBRSP (CNT%, SPR%)
500        WEND
510        '
520        ' ---- CHECK EVENT STATUS BIT ----
530        IF NOT SPR% AND 32 GOTO 750
540           PRINT "  Event status bit set"
550           WRT$ = "*esr?"
560           CALL IBWRT (CNT%, WRT$)
570           ESR$ = SPACE$(255)
580           CALL IBRSP (CNT%, SPR%)
590           CALL IBRD (CNT%, ESR$)
600           ESR% = VAL(ESR$)
610           IF ESR% AND 32 THEN PRINT "    Command error"
620           IF ESR% AND 16 THEN PRINT "    Execution error"
630           IF ESR% AND 4  THEN PRINT "    Query error"
640           '
650           ' ---- READ ERROR MESSAGES ----
660           WRT$ = "syst:err?"
670           ERRMESS$ = SPACE$(255)
680           CALL IBWRT (CNT%, WRT$)
690           CALL IBRD (CNT%, ERRMESS$)
700           WHILE NOT INSTR(ERRMESS$, "No error") <> 0
710              PRINT LEFT$(ERRMESS$, INSTR(ERRMESS$, CHR$(10)))
720              CALL IBWRT (CNT%, WRT$)
730              CALL IBRD (CNT%, ERRMESS$)
740           WEND
750 WEND
760 PRINT "PROGRAM TERMINATED"
770 END
```

This page is intentionally left blank.

'C' examples for National Instruments PC-IIA

Example 1, Statistics

In this example, the counter makes 10000 measurements and uses the statistical functions to determine MAX, MIN, MEAN, and Standard Deviation. All four results are sent to the controller.

```
#include "decl.h"

#include <stdio.h>
#include <process.h>

main ()

{
  int                              Counter, Status, i;
  char                             InString[80];

  Counter = ibfind("DEV10");

  ibwrt(Counter, "*CLS; *SRE 16", 13);
  ibwrt(Counter, "*RST; :FUNC 'Freq 1'; :ACQ:APER 800E-9", 38);
  ibwrt(Counter, ":INP:LEV:AUTO OFF", 17);

  /*Enable statistics on 10000 measurements*/
  ibwrt(Counter, ":CALC:AVER:STAT ON; COUN 10000", 30);
  ibwrt(Counter, ":TRIG:COUN 10000", 16);

  /*Start measurement*/
  ibwrt(Counter, ":Init; *OPC?", 12);

  /*Wait for operation complete (MAV)*/
  printf("Waiting for measurement to get ready\n");
  ibwait(Counter, RQS);

  /**Read status and response**/
  ibrsp(Counter, &Status);
  ibrd(Counter, InString, 80);

  /*Read maximum value**/
  ibwrt(Counter, ":CALC:AVER:TYPE MAX; :CALC:IMM?", 31);
```

```
  ibrd(Counter, InString, 80);
  InString[ibcnt] = '\0';
  printf("Maximum = %s\n", InString);

  /*Read minimum value*/
  ibwrt(Counter, ":CALC:AVER:TYPE MIN; :CALC:IMM?", 31);
  ibrd(Counter, InString, 80);
  InString[ibcnt] = '\0';
  printf("Minimum = %s\n", InString);

  /*Read mean value*/
  ibwrt(Counter, ":CALC:AVER:TYPE MEAN; :CALC:IMM?", 32);
  ibrd(Counter, InString, 80);
  InString[ibcnt] = '\0';
  printf("Mean = %s\n", InString);

  /*Read standard deviation value*/
  ibwrt(Counter, ":CALC:AVER:TYPE SDEV; :CALC:IMM?", 32);
  ibrd(Counter, InString, 80);
  InString[ibcnt] = '\0';
  printf("Standard deviation = %s\n", InString);

  exit(0);
}
```

Example 2, Limit testing

This program uses limit testing to check that the frequency is above a preset value.

```
 #include "decl.h"

#include <stdio.h>
#include <process.h>

main ()

{
  int                              Counter, Status, i;
  char                             InString[80];

  Counter = ibfind("DEV10");

  /*Set continuous frequency measurement*/
  ibwrt(Counter, "*RST; *CLS; :FUNC 'Freq 1'; :INIT:CONT ON", 41);

  /*Enable limit monitoring, limit 1 MHz*/
  ibwrt(Counter, ":CALC:LIM ON; LIM:UPP 1E6; UPP:STAT ON", 38);
  ibwrt(Counter, ":STAT:DREG0:ENAB 2; *SRE 1", 26);

  /*Wait until the limit is passed*/
  printf("Waiting for limit to be passed\n");
  ibwait(Counter, RQS);

  /**Read status and device status register**/
  ibrsp(Counter, &Status);
  ibwrt(Counter, ":STAT:DREG0:EVEN?", 17);
  ibrd(Counter, InString, 80);

  /*Read frequency**/
  ibwrt(Counter, "READ?", 5);
  ibrd(Counter, InString, 80);
  InString[ibcnt] = '\0';
  printf("Frequency = %s\n", InString);

  /*Disable continuous measurement*/
```

```
  ibwrt(Counter, ":INIT:CONT OFF", 14);

  exit(0);
}
```

Example 3, REAL data format

This program uses the REAL data format to speed up the measurement.

/* IEEE 488.2 binary real format follows the 'little-indian' format with the most significant byte first and the least significant byte last. Intel processors use the 'big-indian' format, with the least significant byte first, so we have to reverse the byte order of the incoming block when running on a PC (Intel processor).

```
#include "decl.h"

#include <stdio.h>
#include <process.h>
#include <conio.h>

main ()

{
  int                              Counter, i;
  char                             InString[80];
  double                           DoubleFreq;

  Counter = ibfind("DEV10");

  /*Make the counter output it's result in real format*/
  ibwrt(Counter, ":FORM REAL", 10);

  /*Make continuous measurements until a key is hit*/
  do {

    /*Make a measurement and read the result*/
    ibwrt(Counter, "READ?", 5);
    ibrd(Counter, InString, 80);

    /*Assign the bytes 3...10 of InString to DoubleFreq bytes 7...0.
    The format of InString is #18******** , where "********"
    represents the value.*/
```

```
    for (i=0; i<8; i++)
      ((unsigned char *)&DoubleFreq)[7-i] = InString[3+i];

    /*Print the result*/
    printf("%le\n", DoubleFreq);

  } while (!kbhit());

  /*Restore ascii output format*/
  ibwrt(Counter, ":FORM ASCII", 11);

  exit(0);
}
```

Example 4, Frequency profiling

Frequency profiling visualizes frequency variations for a certain time. This program gives an output file called: PROFILE.DAT. If this file is imported to a spreadsheet program, for instance Lotus 123, you can create a graph like in the figure below.

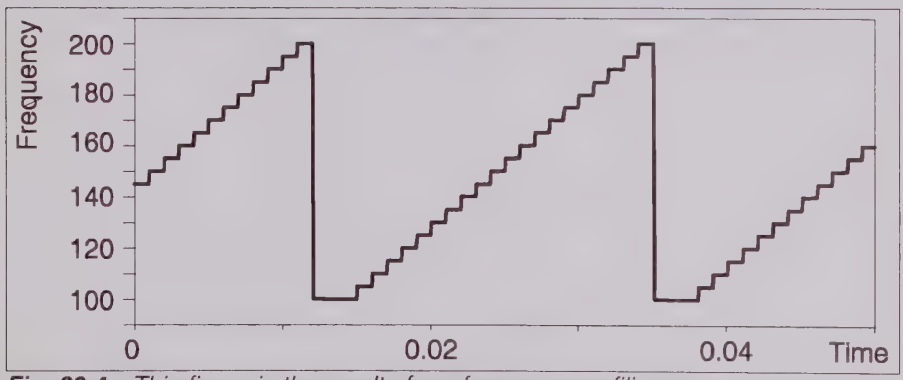


Fig. 20-4 *This figure is the results from frequency profiling on a sweep generator.*

```
#include "decl.h"

#include <stdio.h>
#include <process.h>
#include <string.h>

main ()

{
  int                          Counter, i;
  char                         ArmString[80], InString[80];
  double                       ArmDelay;
  FILE                         *ofp;

  if (ofp = fopen("PROFILE.DAT", "w")) {

    Counter = ibfind("DEV10");
```

```
    /*Enable arming, etc.*/
    ibwrt(Counter, ":TRIG:COUN 1; :ARM:COUN 1; SOUR EXT2", 36);
    ibwrt(Counter, ":INP:LEV:AUTO ONCE; :INP2:LEV:AUTO ONCE", 39);
    ibwrt(Counter, ":DISP:ENAB OFF; :ACQ:APER 1E-6", 30);

    ArmDelay=200e-9;

    /*CAPTURE PROFILE*/

    printf("Profiling");

    for (i=0; i<1000; i++) {
      /*Set arming delay time*/
      sprintf(ArmString, ":ARM:DEL %le", ArmDelay);
      ibwrt(Counter, ArmString, strlen(ArmString));

      /*Measure and read result*/
      ibwrt(Counter, "READ?", 5);
      ibrd(Counter, InString, 80);
      InString[ibcnt] = '\0';

      /*Write arming delay time and result to file*/
      fprintf(ofp, "%le, %s", ArmDelay, InString);

      /*Increase arming delay*/
      ArmDelay += 100e-9;
    }

    ibwrt(Counter, ":DISP:ENAB ON", 13);

    /*Close file*/
    fclose(ofp);

  } else
    printf("CANT OPEN FILE");

  exit(0);
}
```

Example 5, Fast sampling

This program makes a quick array measurement and stores the results in the internal memory of the counter. Then it writes the results to a file called MEAS.DAT. The measurement results as a function of the samples can be visualized in a spreadsheet program like Lotus 123 etc.

```
#include "decl.h"

#include <stdio.h>
#include <process.h>
#include <string.h>

main ()

{
  int                              Counter, Status, i;
  char                             InString[80];
  FILE                             *ofp;

  if (ofp = fopen("MEAS.DAT", "w")) {

    Counter = ibfind("DEV10");

    /*Clear status*/
    ibwrt(Counter, "*CLS", 4);

    /*Enable 1000 measurement with maximum speed*/
    ibwrt(Counter, ":TRIG:COUN 1000; :ARM:COUN 1", 28);
    ibwrt(Counter, ":INP:LEV:AUTO ONCE; :CAL:INT:AUTO OFF", 37);
    ibwrt(Counter, ":DISP:ENAB OFF; :INT:FORM PACKED", 32);
    ibwrt(Counter, ":ACQ:APER 800E-9; :AVER:STAT OFF", 32);

    /**Enable SRQ on operation complete**/
    ibwrt(Counter, "*ESE 1; *SRE 32", 15);

    /*Start measurement*/
    printf("Measuring\n");
    ibwrt(Counter, "INIT; *OPC", 10);
```

```
    /*Wait for operation complete*/
    ibwait(Counter, RQS);

    /**Read status and event status register**/
    ibrsp(Counter, &Status);
    ibwrt(Counter, "*ESR?", 5);
    ibrd(Counter, InString, 80);

    printf("Fetching result");

    for (i=0; i<1000; i++) {
      /*Fetch one result*/
      ibwrt(Counter, "FETCH?", 6);
      ibrd(Counter, InString, 80);
      InString[ibcnt] = '\0';

      /*Write result to file*/
      fprintf(ofp, "%d, %s", i, InString);
    }

    ibwrt(Counter, ":DISP:ENAB ON", 13);

    /*Close file*/
    fclose(ofp);

  } else
    printf("CANT OPEN FILE");

  exit(0);
}
```

This page is intentional left blank.

Chapter 21

Error Messages

The error numbers in the table below are standard SCPI error numbers used in PM 6680.

Command errors		
Error Number	**Error Description**	**description/explanation/examples**
0	No error	
–100	Command error	This is the generic syntax error for devices that cannot detect more specific errors. This code indicates only that a Command Error defined in IEEE-488.2, 11.5.1.1.4 has occurred.
–101	Invalid character	A syntactic element contains a character which is invalid for that type; for example, a header containing an ampersand, SETUP&. This error might be used in place of errors –114, –121, –141, and perhaps some others.
–102	Syntax error	An unrecognized command or data type was encountered; for example, a string was received when the timer/counter does not accept strings.
	Syntax error; unrecognized data	
–103	Invalid separator	The parser was expecting a separator and encountered an illegal character; for example, the semicolon was omitted after a program message unit, ✱EMC1:CH1:VOLTS5.
–104	Data type error	The parser recognized a data element different than one allowed; for example, numeric or string data was expected but block data was encountered.
–105	GET not allowed	A Group Execute Trigger was received within a program message (see IEEE-488.2, 7.7).
–108	Parameter not allowed	More parameters were received than expected for the header; for example, the ✱EMC common command accepts only one parameter, so receiving ✱EMC0,,1 is not allowed.

Command errors		
Error Number	**Error Description**	**description/explanation/examples**
–109	Missing parameter	Fewer parameters were received than required for the header; for example, the ✷EMC common command requires one parameter, so receiving ✷EMC is not allowed.
–110	Command header error	An error was detected in the header. This error message is used when the timer/counter cannot detect the more specific errors described for errors –111 though –119.
–111	Header separator error	A character which is not a legal header separator was encountered while parsing the header; for example, no space followed the header, thus ✷GMC"MACRO" is an error.
–112	Program mnemonic too long	The header contains more than 12 characters (see IEEE-488.2, 7.6.1.4.1).
–113	Undefined header	The header is syntactically correct, but it is undefined for this specific timer/counter; for example, ✷XYZ is not defined for any device.
–114	Header suffix out of range	Indicates that a non-header character has been encountered in what the parser expects is a header element.
–120	Numeric data error	This error, as well as errors –121 through –129, are generated when parsing a data element which appears to be of a numeric type. This particular error message is used when the timer/counter cannot detect a more specific error.
	Numeric data error; overflow from conversion	
	Numeric data error; underflow from conversion	
	Numeric data error; not a number from conversion	

Command errors		
Error Number	**Error Description**	**description/explanation/examples**
–121	Invalid character in number	An invalid character for the data type being parsed was encountered; for example, an alpha in a decimal numeric or a "0" in octal data.
–123	Exponent too large	The magnitude of the exponent was larger than 32000 (see IEEE-488.2, 7.7.2.4.1).
–124	Too many digits	The mantissa of a decimal numeric data element contained more than 255 digits excluding leading zeros (see IEEE-488.2, 7.7.2.4.1).
–128	Numeric data not allowed	A legal numeric data element was received, but the timer/counter does not accept it in this position for the header.
–130	Suffix error	This error as well as errors –131 through –139 is generated when parsing a suffix. This particular error message is used when the timer/counter cannot detect a more specific error.
–131	Invalid suffix	The suffix does not follow the syntax described in IEEE-488.2, 7.7.3.2, or the suffix is inappropriate for this timer/counter.
–134	Suffix too long	The suffix contained more than 12 characters (see IEEE-488.2, 7.7.3.4).
–138	Suffix not allowed	A suffix was encountered after a numeric element which does not allow suffixes.
–140	Character data error	This error as well as errors 141 through –149 is generated when parsing a character data element. This particular error message is used when the timer/counter cannot detect a more specific error.
–141	Invalid character data	Either the character data element contains an invalid character or the particular element received is not valid for the header.
–144	Character data too long	The character data element contains more than 12 characters (see IEEE-488.2, 7.7.1.4).

Command errors		
Error Number	**Error Description**	**description/explanation/examples**
–148	Character data not allowed	A legal character data element was encountered where prohibited by the timer/counter.
–150	String data error	This error as well as errors –151 through –159 is generated when parsing a string data element. This particular error message is used when the timer/counter cannot detect a more specific error.
–151	Invalid string data Invalid string data; unexpected end of message	A string data element was expected, but was invalid for some reason (see IEEE-488.2, 7.7.5.2); for example, an END message was received before the terminal quote character.
–158	String data not allowed	A string data element was encountered but was not allowed by PM 6680 at this point in parsing.
–160	Block data error	This error as well as errors –161 through –169 is generated when parsing a block data element. This particular error message is used when PM 6680 cannot detect a more specific error.
–161	Invalid block data	A block data element was expected, but was invalid for some reason (see IEEE-488.2, 7.7.6.2); for example, an END message was received before the length was satisfied.
–168	Block data not allowed	A legal block data element was encountered but was not allowed by the timer/counter at this point in parsing.

Command errors		
Error Number	**Error Description**	**description/explanation/examples**
–170	Expression data error	This error as well as errors –171 through –179 is generated when parsing an expression data element. This particular error message is used if the timer/counter cannot detect a more specific error.
	Expression data error; floating point underflow	The floating point operations specified in the expression caused a floating point error.
	Expression data error; floating point overflow	
	Expression data error; not a number	
	Expression data error; different number of channels given	Two channel list specifications, giving primary and secondary channels for 2-channel measurements, contained a different number of channels.
–171	Invalid expression data	The expression data element was invalid (see IEEE-488.2, 7.7.7.2); for example, unmatched parentheses or an illegal character were used.
	Invalid expression data; bad mnemonic	A mnemonic data element in the expression was not valid.
	Invalid expression data; illegal element	The expression contained a hexadecimal element not permitted in expressions.
	Invalid expression data; unexpected end of message	End of message occurred before the closing parenthesis.
	Invalid expression data; unrecognized expression type	The expression could not be recognized as either a mathematical expression, a data element list or a channel list.
–178	Expression data not allowed	A legal expression data was encountered but was not allowed by the timer/counter at this point in parsing.

Command errors		
Error Number	**Error Description**	**description/explanation/examples**
–180	Macro error	This error as well as errors –181 through –189 is generated when defining a macro or executing a macro. This particular error message is used when the timer/counter cannot detect a more specific error.
–181	Invalid outside macro definition	Indicates that a macro parameter placeholder ($<number) was encountered outside of a macro definition.
–183	Invalid inside macro definition	Indicates that the program message unit sequence, sent with a *DDT or *DMC command, is syntactically invalid (see IEEE-10.7.6.3).
–184	Macro parameter error	Indicates that a command inside the macro definition had the wrong number or type of parameters.
	Macro parameter error; unused parameter	The parameter numbers given are not continuous; one or more numbers have been skipped.
	Macro parameter error; badly formed placeholder	The'$' sign was not followed by a single digit between 1 and 9.
	Macro parameter error; parameter count mismatch	The macro was invoked with a different number of parameters than used in the definition.

Execution errors		
Error Number	**Error Description**	**description/explanation/examples**
–200	Execution error	This is the generic syntax error for devices that cannot detect more specific errors. This code indicates only that an Execution Error as defined in IEEE-488.2, 11.5.1.1.5 has occurred.
–210	Trigger error	

Execution errors		
Error Number	**Error Description**	**description/explanation/examples**
–211	Trigger ignored	Indicates that a GET, ✱TRG, or triggering signal was received and recognized by the timer/counter but was ignored because of timer/counter timing considerations; for example, the timer/counter was not ready to respond.
–212	Arm ignored	Indicates that an arming signal was received and recognized by the timer/counter but was ignored.
–213	Init ignored	Indicates that a request for a measurement initiation was ignored because another measurement was already in progress.
–214	Trigger deadlock	Indicates that the trigger source for the initiation of a measurement is set to GET and subsequent measurement query is received. The measurement cannot be started until a GET is received, but the GET would cause an INTERRUPTED error.
–215	Arm deadlock	Indicates that the arm source for the initiation of a measurement is set to GET and subsequent measurement query is received. The measurement cannot be started until a GET is received, but the GET would cause an INTERRUPTED error.
–220	Parameter error	Indicates that a program data element related error occurred. This error message is used when the timer/counter cannot detect the more specific errors –221 to –229.

Execution errors		
Error Number	**Error Description**	**description/explanation/examples**
–221	Settings conflict	Indicates that a legal program data element was parsed but could not be executed due to the current timer/counter state (see IEEE-488.2, 6.4.5.3 and 11.5.1.1.5.)
	Settings conflict; PUD memory is protected	
	Settings conflict; invalid combination of channel and function	
–222	Data out of range	Indicates that a legal program data element was parsed but could not be executed because the interpreted value was outside the legal range as defined by the timer/counter (see IEEE-488.2, 11.5.1.1.5.)
	Data out of range; exponent too large	The expression was too large for the internal floating point format
	Data out of range; below minimum	Data below minimum for this function/parameter.
	Data out of range; above maximum	Data above maximum for this function/ parameter.
	Data out of range; requires attenuator on	The trig level given can only be set with input attenuator set to 10X
	Data out of range; (Save/recall memory number)	A number outside 0 to 19 was given for the save/recall memory
–223	Too much data	Indicates that a legal program data element of block, expression, or string type received that contained more data than the timer/counter could handle due to memory or related timer/counter-specific requirements.
	Too much data; ✱PUD string too long	
	Too much data; ✱DDT definition too long	

Execution errors		
Error Number	**Error Description**	**description/explanation/examples**
–224	Illegal parameter value	Used where exact value, from a list of possible values, was expected.
–230	Data corrupt or stale	Possibly invalid data; new reading started but not completed since last access.
–231	Data questionable	
	Data questionable; one or more data elements ignored	One or more data elements sent with a MEASure or CONFigure command was ignored by the timer/counter.
–240	Hardware error	Indicates that a legal program command or query could not be executed because of a hardware problem in the timer/counter. Definition of what constitutes a hardware problem is completely device specific. This error message is used when the timer/counter cannot detect the more specific errors described for errors –241 through –249.
–241	Hardware missing	Indicates that a legal program command or query could not be executed because of missing timer/counter hardware; for example, an option was not installed. Definition of what constitutes missing hardware is completely device specific.
	Hardware missing; (prescaler)"	
–254	Media full	Indicates that a legal program command or query could not be executed because the media was full; for example, there is no room on the disk. The definition of what constitutes a full media is device specific.
–258	Media protected	Indicates that a legal program command or query could not be executed because the media was protected; for example, the write-protect tab on a disk was present. The definition of what constitutes protected media is device specific.

Execution errors		
Error Number	**Error Description**	**description/explanation/examples**
–260	Expression error	Indicates that a expression program data element related error occurred. This error message is used when the timer/counter cannot detect the more specific errors described for errors –261 through –269.
–261	Math error in expression	Indicates that a syntactically correct expression program data element could not be executed due to a math error; for example, a divide-by-zero was attempted.
–270	Macro error	Indicates that a macro-related execution error occurred. This error message is used when the timer/counter cannot detect the more specific error described for errors –271 through –279.
	Macro error; out of name space	No room for any more macro names.
	Macro error; out of definition space	No room for this macro definition.
–271	Macro syntax error	Indicates that a syntactically correct macro program data sequence, according to IEEE-488.2 10.7.2, could not be executed due to a syntax error within the macro definition (see IEEE-488.2, 10.7.6.3)
–272	Macro execution error	Indicates that a syntactically correct macro program data sequence could not be executed due to some error in the macro definition (see IEEE-488.2, 10.7.6.3)
–273	Illegal macro label	Indicates that the macro label defined in the *DMC command was a legal string syntax, but could not be accepted by the timer/counter (see IEEE-488.2, 10.7.3 and 10.7.6.2); for example, the label was too long, the same as a common command header, or contained invalid header syntax.

Execution errors		
Error Number	**Error Description**	**description/explanation/examples**
–274	Macro parameter error	Indicates that the macro definition improperly used a macro parameter place holder (see IEEE-488.2, 10.7.3).
–275	Macro definition too long	Indicates that a syntactically correct macro program data sequence could not be executed because the string or block contents were too long for the timer/counter to handle (see IEEE-488.2, 10.7.6.1).
–276	Macro recursion error	Indicates that a syntactically correct macro program data sequence could not be executed because the timer/counter found it to be recursive (see IEEE-488.2, 10.7.6.6).
–277	Macro redefinition not allowed	Indicates that a syntactically correct macro label in the ✱DMC command could not be executed because the macro label was already defined (see IEEE-488.2, 10.7.6.4).
–278	Macro header not found	Indicates that a syntactically correct macro label in the ✱GMC? query could not be executed because the header was not previously defined.

Standardized Device specific errors		
Error Number	**Error Description**	**description/explanation/examples**
–300	Device specific error	This code indicates only that a Device-Dependent Error as defined in IEEE-488.2, 11.5.1.1.6 has occurred. Contact your local Fluke/Philips Service center.
–311	Memory error	Indicates that an error was detected in the timer/counter's memory. Contact your local Fluke/Philips Service center.
–312	PUD memory lost	Indicates that the protected user data saved by the ✱PUD command has been lost. Contact your local Fluke/Philips Service center.

Standardized Device specific errors		
Error Number	**Error Description**	**description/explanation/examples**
–314	Save/recall memory lost	Indicates that the nonvolatile calibration data used by the *SAV? command has been lost. Contact your local Fluke/Philips Service center.
–330	Self-test failed	Contact your local Fluke/Philips Service center.
–350	Queue overflow	A specific code entered into the queue in lieu of the code that caused the error. This code indicates that there is no room in the queue and an error occurred but was not recorded.

Query errors		
Error Number	**Error Description**	**description/explanation/examples**
–400	Query error	This code indicates only that a Query Error as defined in IEEE-488.2, 11.5.1.1.7 and 6.3 has occurred.
–410	Query INTERRUPTED	Indicates that a condition causing an INTERRUPTED Query error occurred (see IEEE-488.2, 6.3.2.3); for example, a query was followed by DAB or GET before a response was completely sent.
	Query INTERRUPTED; in send state	The additional information indicates the IEEE-488.2 message exchange state where the error occurred.
	Query INTERRUPTED; in query state	
	Query INTERRUPTED; in response state	

Query errors		
Error Number	**Error Description**	**description/explanation/examples**
–420	Query UNTERMINATED	Indicates that a condition causing an UNTERMINATED Query error occurred (see IEEE-488.2, 6.3.2.2); for example, the timer/counter was addressed to talk and an incomplete program message was received.
	Query UNTERMINATED; in idle state	The additional information indicates the IEEE-488.2 message exchange state where the error occurred
	Query UNTERMINATED; in read state	
	Query UNTERMINATED; in send state	
–430	Query DEADLOCKED	Indicates that a condition causing an DEADLOCKED Query error occurred (see IEEE-488.2, 6.3.1.7); for example, both input buffer and output buffer are full and the timer/counter cannot continue.
–440	Query UNTERMINATED after indefinite response	Indicates that a query was received in the same program message after an query requesting an indefinite response was executed (see IEEE-488.2, 6.5.7.5.7.)

PM 6680 Device specific errors		
Error Number	**Error Description**	**description/explanation/examples**
100	Device operation gave floating point underflow	A floating point error occurred during a timer/counter operation.
101	Device operation gave floating point overflow	A floating point error occurred during a timer/counter operation.
102	Device operation gave 'not a number'	A floating point error occurred during a timer/counter operation.
110	Invalid measurement function	The timer/counter was requested to set a measurement function it could not make.
120	Save/recall memory protected	An attempt was made to write in a protected memory.
130	Unsupported command	Indicates a mismatch between bus and timer/counter capabilities.
131	Unsupported boolean command	
132	Unsupported decimal command	
133	Unsupported enumerated command	
134	Unsupported auto command	
135	Unsupported single shot command	
136	Command queue full; last command discarded	The timer/counter has an internal command queue with room for about 100 commands. A large number of commands arrived fast without any intervening query.
137	Inappropriate suffix unit	A suffix unit was not appropriate for the command. Recognized units are Hz (Hertz), s (seconds), Ohm (Ω) and V (Volt).

PM 6680 Device specific errors		
Error Number	**Error Description**	**description/explanation/examples**
138	Unexpected command to device execution	A command reached timer/counter execution which should have been handled by the bus.
139	Unexpected query to device execution	A query reached timer/counter execution which should have been handled by the bus.
150	Bad math expression format	Only a fixed, specific math expression is recognized by the timer/counter, and this was not it.
160	Measurement broken off	A new bus command caused a running measurement to be broken off.
170	Instrument set to default	An internal setting inconsistency caused the instrument to go to default setting.
200	Message exchange error	An error occurred in the message exchange handler (generic error).
201	Reset during bus input	The instrument was waiting for more bus input, but the waiting was broken by the operator.
202	Reset during bus output	The instrument was waiting for more bus output to be read, but the waiting was broken by the operator.
203	Bad message exchange control state	An internal error in the message exchange handler.
204	Unexpected reason for GPIB interrupt	A spurious GPIB interrupt occurred, not conforming to any valid reason like an incoming byte, address change, etc.
205	No listener on bus when trying to respond	This error is generated when the timer/counter is an active talker, and tries to send a byte on the bus, but there are no active listeners. (This may occur if the controller issues the device talker address before its own listener address, which some PC controller cards has been known to do)
210	Mnemonic table error	An abnormal condition occurred in connection with the mnemonics tables (generic error).
211	Wrong macro table checksum found	The macro definitions have been corrupted (could be loss of memory).

PM 6680 Device specific errors		
Error Number	**Error Description**	**description/explanation/examples**
212	Wrong hash table checksum found	The hash table has been corrupted. Could be bad memory chips or address logic. Contact your local Fluke/Philips Service center.
213	RAM failure to hold information (hash table)	The memory did not retain information written to it. Could be bad memory chips or address logic. Contact your local Fluke/Philips Service center.
214	Hash table overflow	The hash table was too small to hold all mnemonics. Ordinarily indicates a failure to read (RAM or ROM) correctly. Contact your local Fluke/Philips Service center.
220	Parser error	Generic error in the parser.
221	Illegal parser call	The parser was called when it should not be active.
222	Unrecognized input character	A character not in the valid IEEE488.2 character set was part of a command.
223	Internal parser error	The parser reached an unexpected internal state.
230	Response formatter error	Generic error in the response formatter.
231	Bad response formatter call	The response formatter was called when it should not be active.
232	Bad response formatter call (eom)	The parser was called to output an end of message, when it should not be active.
233	Invalid function code to response formatter	The response formatter was requested to output data for an unrecognized function.
234	Invalid header type to response formatter	The response formatter was called with bad data for the response header (normally empty)
235	Invalid data type to response formatter	The response formatter was called with bad data for the response data.
240	Unrecognized error number in error queue	An error number was found in the error queue for which no matching error information was found.

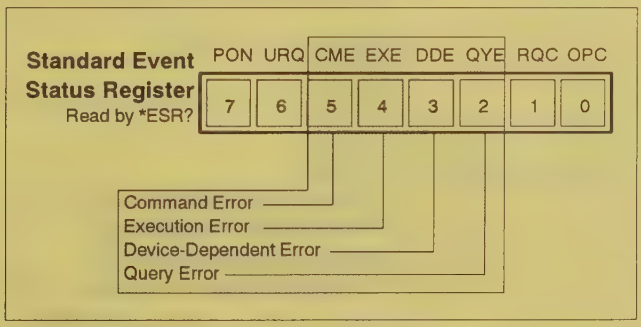
This page is intentionally left blank.

How to use Error Reporting

The PM 6680 will place a detected error in its Error Queue. This queue is a FIFO (First-In First-Out) buffer. When you read the queue, the first error will come out first, the last error last.

- *How to detect the presence of errors in the queue*

Bit 2 in the Status Byte Register, if set true, shows that the instrument has detected errors. It is also possible to enable this bit for Service Request on the GPIB. This can then interrupt the GPIB controller program on occurrence of an error.



21-1 *Error bits in the Standard Event Status Register*

How to read the Error/Event Queue

This is done with the :SYSTem:ERRor? query.

Example:

SEND -> **`:SYSTem:ERRor?`**

READ <- **`–100, "Command Error"`**

The query returns the error number followed by the error description.

If more than one error occurred, the query will return the error that occurred first. When you read an error you will also remove it from the queue. You can read the next error by repeating the query. When you have read all errors the queue is empty, and the :SYSTem:ERRor? query will return:

0, "No error"

When errors occur and you do not read these errors, the Error Queue may overflow. Then the instrument will overwrite the last error in the queue with:

–350, "Queue overflow"

If more errors occur they will be discarded.

Standardized Error numbers

The instrument reports four classes of standardized errors in the Standard Event Status and in the Error/Event Queue as shown in the following table:

Error class	Range of Error numbers	Standard Event Register
Command Error	–100 to –199	bit 5 - CME
Execution Error	–200 to –299	bit 4 - EXE
Device specific Error	–300 to –399 +100 to +32767•	bit 3 - DDE
Query Error	–400 to –499	bit 2 - QYE

- *Command Error* — This error shows that the instrument detected a syntax error.
- *Execution Error* — This error shows that the instrument has received a valid program message which it cannot execute because of some device specific conditions.
- *Device specific Error* — This error shows that the instrument could not properly complete some device specific operations.
- *Query Error* — This error will occur when the Message Exchange Protocol is violated, for example, when you send a query to the instrument and then send a new command without first reading the response data from the previous query. Also, trying to read

data from the instrument without first sending a query to the instrument will cause this error.

This page is intentionally left blank.

Chapter 22

Appendix

This page is intentionally left blank.

Appendix 1, Alphanumeric Command List

IEEE-488.2 Common Commands

✱**CLS** The ✱CLS common command clears the status data structures by clearing all event registers and the error queue.

✱**DMC**␣<macro label>,<program messages> Defines a macro called<macro label> which performs the actions defined by <program messages>

✱**EMC**␣1|0 Enabled/disables the execution of macros.

✱**ESE** Sets the enable bits of the standard event enable register.

✱**ESR?** Reads out the contents of the standard event status register.

✱**GMC?**␣**<macro label>** The counter responds with the definition for <macro label>

✱**IDN?** Reads out the manufacturer, model, serial number, Frimware level for main and GPIB program.

✱**LMC?** The counter responds with the list of all current <macro labels>

✱**LRN?** Causes the counter to respond with its current setting.

✱**OPC?** Places an ASCII character 1 into the device's Output Queue when all pending selected device operations have been finished.

✱**OPC** Causes the device to generate the operation complete message in the Standard Event Status Register when all pending selected device operations have been finished.

✱**OPT?** Response is a list of all detectable options.

✱**PMC** Removes all macros.

✱**PSC** Enables/disables automatic power-on clearing.

✱**PUD**␣**<arbitrary block program data>** arbitrary block program data is 64 bytes of protected user data, can only be written after unprotect.

✱**RCL**␣**<memory position>** Recalls setting No. <memory position>.

✱RST Reset

✱**SAV**␣**<memory position>** Saves the current setting of the instrument in <memory position>

✱**SRE** Sets the service request enable register bits.

✱**STB?** Reads out the value of the service request enable register.

✱**TRG** Device-specific equivalent of the IEEE 488.1 defined Group Execute Trigger.

***TST?** Starts internal self-test and responds: 0=No error, 1=RAM Failure, 2=ROM 1 Failure, 4=Logic Failure, 8=Display Failure.

***WAI** Prevents the device from executing any further commands or queries until execution of all previous commands or queries have been completed.

A

:ABORt Terminates a measurement.

:ACQuisition:HOFF:ECOunt␣<hold off event value>|MIN|MAX Sets the hold off event value. The counter counts negative events on the B-input (channel 2).

:ACQuisition:HOFF:MODE␣TIME|EVENT Selects if triggering is going to be disabled for a preset time or for a preset number of events

:ACQuisition:HOFF:TIME␣<hold off time value>|MIN|MAX Sets the hold-off time value.

:ACQuisition:HOFF␣<boolean> Switches the hold off function on/off

:ACQuisition:RESolution␣HIGH|LOW Selects basic measurement mode for all time related measurements. HIGH = 0.5ns, LOW = 100ns.

:ARM:COUNt␣<numeric value>|MIN|MAX No. of Measurements on each Bus arm.

:ARM:DELay␣<numeric value>|MIN|MAX Delay after External Start Arming.

:ARM:ECOunt␣<numeric value>|MIN|MAX External Events before Start Arming.

:ARM:LAYer2:SOURce␣BUS|IMMediate Switches between Bus and Immediate mode for the "wait-for-bus-arm" function.

:ARM:LAYer2 Bus Arming Exit, overrides waiting for bus arm, provided the source is set to bus.

:ARM:SLOPe␣POSitive|NEGative External Arming Start Slope.

:ARM:SOURce␣EXTernal2|EXTernal4|IMMediate External Arming Start Source.

:ARM:STOP:DELay␣<numeric value> Delay after External Stop Arming

:ARM:STOP:SLOPe␣POSitive|NEGative External Stop Arming Slope.

:ARM:STOP:SOURce␣EXTernal2|EXTernal4|IMMediate External Stop Arming Source.

:ARM:STOP:ECOunt␣<numeric value> External Events before Stop Arming.

:AVERage:COUNt␣<numeric value>|MIN|MAX Sets number of samples for time interval averaging measurements. (e.g. PWIDTH, TIME, RISE and FALL.)

:AVERage:MODE␣TIME|COUNts Selects if the gate time (:SENS:ACQ:APER) or a preset number of events (:SENS:AVER:COUN) is going to limit the No. of samples when doing time interval averaging measurements (e.g., PWID, TIME)

:AVERage:STATe␣<boolean> Switch on/off the average function.

C

:CALCulate:AVERage:COUNt␣<numeric value>|MAX|MIN Selects sample size for STAT.

:CALCulate:AVERage:TYPE␣MAX|MIN|MEAN|SDEViation Select which STAT function should be performed.

:CALCulate:AVERage␣<boolean> Switches on/off the STAT function.

:CALCulate:DATA? Reads out data from the statistical functions.

:CALCulate:IMMediate Makes new statistical calculations on samples already taken.

:CALCulate:LIMit:FAIL? Returns 1 if the measurement result has passed the limit.

:CALCulate:LIMit:LOWer␣ Set lower limit.

:CALCulate:LIMit:LOWer:STATe␣ Check against lower limit on/off.

:CALCulate:LIMit:UPPer:STATe␣<boolean> Check against upper limit on/off.

:CALCulate:LIMit:UPPer␣ Sets the value of the upper limit.

:CALCulate:LIMit␣<boolean> Switches on/off monitoring of parameter limits.

:CALCulate:MATH:STATe␣<boolean> Switches on/off the MATH function.

:CALCulate:MATH␣<numeric expression> Defines the expression for math. Use ((K*X)+L)/M or ((K/X)+L)/M.

:CALCulate:STATe␣<boolean> Switches on and off the complete post processing block (mathematics, statistics, averaging and limit monitoring)

:CALibration:INTerpolator:AUTO␣<boolean>|ONCE Switches on/off auto-calibration of interpolators.

:CONFigure:ARRay<Measuring Function>␣(<Array Size>),<Parameters>,(<Channels>) Setup Instrument for an array Measurement with <array size> No. of measurements.

:CONFigure<Measuring Function>␣<Parameters>,(<Channels>) Setup Instrument for Measurement

---

D

:DCYCle␣[<threshold>][,(@1|2|4)] Measures the positive duty cycle.

:DISPlay:ENABle␣<boolean> Switches on/off the updating of the display.

F

:FALL:TIME ␣ Measures the fall time on input A.

:FETCh:ARRay?␣<Array Size>|MAX Fetch a array of Measurement Data

:FETCh? Fetch Measurement Data

:FORMat:FIXed␣<boolean> Selects fixed or floating ASCII format.

:FORMat␣ASCii|REAL Selects response data type.

:FREQuency:BURSt␣[<expected value>[,<resolution>]][,(@1|2|3|4)] Measures the CW frequency on the input specified in the channel list.

:FREQuency:PRFrequency␣[<expected value>[,<resolution>]][,(@1|2|3|4)] Measures the PRF on the input specified in the channel list.

:FREQuency:RANGe:LOWer␣<numeric value>|MIN|MAX High Speed Voltage Measurements. MAX gives faster voltage and autotrigger on signals above 10kHz.

:FREQuency:RATio␣[,(@1|2|3|4),(@1|2|3|4)] Measures frequency ratio between the two inputs specified in the channel list.

:FREQuency␣[<expected value>[,<resolution>]][,(@1|2|3|4|5|6|7|)] Measures frequency on the input specified in the channel list.

:FUNCtion␣'FREQuency:PRF␣1|2|3|4|5|6|7 Sets up the counter for a PRF frequency measurement.

:FUNCtion␣'TINTerval ␣1|2|4,1|2|4' Sets up the counter for a time interval measurement.

:FUNCtion␣'FREQuency:RATio␣1|2|3|4|5,1|2|3|4|5' Sets up the counter for a frequency ratio measurement.

:FUNCtion␣'FREQuency␣1|2|3|4|5|6|7' Sets up the counter for a frequency measurement.

:FUNCtion␣'PHASe␣1|2,1|2' Sets up the counter for a phase measurement.

:FUNCtion␣'TOTalize:GATed␣1|2|4,1|2|4' Sets up the counter for totalizing on primary channel, gated by the signal on the secondary channel.

:FUNCtion␣'NDUTycycle␣1|2|4' Sets up the counter for a negative duty cycle measurement.

:FUNCtion␣'PWIDth␣1|2|4' Sets up the counter for a positive pulse width measurement.

:FUNCtion␣'NWIDth␣1|2|4' Sets up the counter for a negative pulse width measurement.

:FUNCtion␣'DCYCle|PDUTycycle␣1|2|4 Sets up the counter for a positive duty cycle measurement.

:FUNCtion␣'FREQuency:BURSt␣1|2|3|4|5|6|7' Sets up the counter for a CW frequency measurement.

:FUNCtion␣'TOTalize:TIMed␣1|2|4' Set up the counter for totalizing on one channel for a preset time.

:FUNCtion␣'VOLTage:MAXimum␣1|2' Sets up the counter for a Voltage max measurement.

:FUNCtion␣'VOLTage:MINimum␣1|2' Sets up the counter for a Voltage min measurement.

:FUNCtion␣'RISE:TIME␣1' Sets up the counter for a rise time measurement on input A.

:FUNCtion␣'FALL:TIME␣1' Sets up the counter for a fall time measurement on input A.

:FUNCtion␣'VOLTage:PTPeak␣1|2' Sets up the counter for a Voltage PTP measurement.

:FUNCtion␣'TOTalize:ACCumulated␣1|2|4,1|2|4' Totalizes on primary channel, gated by the signal on the secondary channel. The result are the sum of all totalizations during <time for gate open>.

:FUNCtion␣'TOTalize:SSTop␣1|2|4,1|2|4' Set up the counter to totalize on the primary channel. The signal on the secondary channel is the start/stop signal.

:FUNCtion␣'TOTalize ␣1|2|4,1|2|4' Sets up the counter for count/totalize controlled from the GPIB interface using the command SENS:TOT:GATE␣ON|OFF.

:FUNCtion␣'PERiod␣1|2|3|4|5|6|7' Sets up the counter for a period measurement.

I

:INITiate:CONTinuous␣<boolean> OFF=Trigger system in "idle-state" until Continuous ON or :INITiate is received. ON= the completion of a measurement cycle immediately starts a new trigger cycle without

:INITiate initiates a measurement.

:INPut:ATTenuation␣<numeric value>|MAX|MIN Sets attenuation to 1X or 10X.

:INPut:COUPling␣AC | DC Selects AC or DC coupling.

:INPut:FILTer␣<boolean> Switches on/off the 100 kHz LP-filter on input A.

:INPut:IMPedance␣<numeric value>|MAX|MIN Sets input impedance to 50Ω or 1MΩ.

:INPut:LEVel:AUTO␣<boolean>|ONCE Switches on/off AUTO-trigger for input A.

:INPut:LEVel␣<numeric value>|MAX|MIN Sets the trigger level.

:INPut:SLOPe␣POS|NEG Selects positive or negative trigger slope.

:INPut2:ATTenuation␣<numeric value>|MAX|MIN Sets attenuation to 1X or 10X.

:INPut2:COMMon␣ <boolean> Internally connects the signal on input A to input B also.

:INPut2:COUPling␣AC | DC Selects AC or DC coupling.

:INPut2:IMPedance␣<numeric value>|MAX|MIN Sets input impedance to 50Ω or 1MΩ.

:INPut2:LEVel␣<numeric value>|MAX|MIN Sets the trigger level.

:INPut2:SLOPe␣POS|NEG Selects positive or negative trigger slope.

:INPut2:AUTO␣<boolean>|ONCE Switches on/off AUTO-trigger for input B.

:INPut4:SLOPe␣POS|NEG Selects positive or negative trigger slope.

:INTernal:FORMat␣REAL|PACKed Selects internal data format of the measurement result from the SENSe block.

M

:MAXimum [␣(@1|@2)] Measures the maximum voltage.

:MEASure:ARRay<Measuring Function>?␣(<Array Size>) Setup the Instrument,Perform an array Measurement with <array size> No. of measurements, and Reads Data

:MEASure<Measuring Function>?␣<Parameters>,(<Channels>) Setup the Instrument, Perform Measurement and Read Data.

:MEMory:DELete:MACRo␣'<name>' Deletes the macro called <name>

:MEMory:FREE:MACRo? Returns the No. of bytes available for macros and the No. of bytes used.

:MEMory:NSTates? Returns the number of available memory positions for *SAV.

:MINimum[␣(@1|@2)] Measures the minimum voltage.

N

:NDUTycycle␣[<threshold>][,(@1|2|4)] Measures the negative duty cycle.

:NWIDth␣[<threshold>][,(@1|2|4)] Measures the negative pulse width.

O

:OUTPut:SCALe␣<scaling factor> Sets the scaling factor of the analog output.

:OUTPut␣<boolean> Switches the analog output on and off.

P

:PDUTycycle␣[<threshold>][,(@1|2|4)] Measures the positive duty cycle.

:PERiod ␣[<expected value>[,<resolution>]][,(@1|2|3|4)] Measures the period of the signal on the input specified in the channel list.

:PHASe␣[[,(@1|2),(@1|2)] Measures the phase between the signals on the channels in the channel list.

:PTPeak [␣(@1|@2)] Measures the peak-to-peak voltage.

:PWIDth ␣[<threshold>][,(@1|2|4)] Measures the positive pulse width.

R

:READ:ARRay?␣<Array Size>|MAX Performs an array Measurement with <array size> No. of measurements and Reads Data

:READ? Performs Measurement and Reads Data

:RISE:TIME Measures the rise time on input A.

:ROSCillator:SOURce␣INTernal|EXTernal Select the signal from the external reference input as clock instead of the internal oscillator.

S

:STATus:DREGister0:ENABle? Reads the enable bit of the Device Register 0.

:STATus:DREGister0:ENABle␣<bit mask> Sets the enable bit of the Device Register 0.

:STATus:DREGister0? Reads out the contents of the Device Event Register.

:STATus:OPERation:CONDition? Reads contents of the operation condition register.

:STATus:OPERation:ENABle␣<bit mask> Sets the enable bits in the operation status enable register.

:STATus:OPERation:ENABle? Outputs the current setting of the enable bits in the operation status enable register.

:STATus:OPERation? Reads the contents of the operation event status register.

:STATus:PRESet Enable Device Status Reporting.

:STATus:QUEStionable:CONDition? Reads out the contents of the status questionable condition register.

:STATus:QUEStionable:ENABle? Reads the enable bits of the status questionable enable register.

:STATus:QUEStionable:ENABle␣<bit mask> Sets the enable bits of the status questionable enable register.

:STATus:QUEStionable? Reads out the contents of the status questionable event register.

:SYSTem:ERRor? The counter responds with an ASCII text description of an error.

:SYSTem:KEY␣ <code>|MIN|MAX Simulates a press on the specified key. <code> is the hardware code for each key. KEY 0 means release key and must be used.

:SYSTem:PRESet Same as pressing the front panel key LOCAL/PRESET

:SYSTem:SET␣ <block data> Transmits the complete current state of the instrument in binary form.

:SYSTem:TOUT:TIME␣<time out value> This command sets the time out in seconds.

:SYSTem:TOUT␣<boolean> Switches on or off the time out.

:SYSTem:UNPRotect Unprotects the user data (set/read by ✱PUD) and front setting memories until next PMT (Program message terminator) or Device clear or Reset (✱RST).

:SYSTem:VERSion? returns the SCPI system version that this instrument complies to.

:SYSTem:COMMunicate:GPIB:ADDRess␣<numeric value>|MIN|MAX Sets the GPIB address, overrides the switches on the rear panel.

T

:TEST:CHECk␣<boolean> Connects the internal 10MHz reference signal to the measuring logic.

:TEST:SELect␣RAM|ROM|LOGic|ALL Selects which internal self-tests shall be used when self-test is requested by the ✱TST command.

:TINTerval␣[,(@1|@2|@4),(@1|@2|@4)] Measures the time interval between the signals on the channels specified in the channel list.

:TOTalize:ACCumulated␣[<time for gate open>][,(@1|@2|@4)[,(@1|@2|@4)]] Totalizes on primary channel, gated by the signal on the secondary channel. The result are the sum of all totalizations during <time for gate open>.

:TOTalize:CONTinuous[␣(@1|@2|@4)[,(@1|@2|@4)]] Adds the total on the primary channel and subtracts total on the secondary channel. The result is the difference between all totalizations when the gate is closed again. Gate is controlled by key on front or :SENS:TOT:GATE ON/OFF. Only for :CONFigure

:TOTalize:GATE␣<boolean> Open/closes the gate for the TOTalize.

:TOTalize:GATed␣ [,(@1|@2|@4)[,(@1|@2|@4)]] Totalizes on primary channel, gated by the signal on the secondary channel.

:TOTalize:SSTop[␣(@1|2|4)[,(@1|2|4)]] Totalizes on primary channel, started and stopped by the signal on the secondary channel.

:TOTalize:TIMed␣[<time for gate open>][,(@1|@2|@4)[,(@1|@2|@4)]] Adds the total on the primary channel and subtracts total on the secondary channel. The result is the difference between all totalizations after <time for gate open>.

:TRIGger:COUNt␣<numeric value>|MIN|MAX No. of Triggerings on each Ext Arm start.

V

:VOLTage:GATed:STATe␣<boolean> Selects the gated mode for the VOLTage:MAX|MIN|PTPeak measuring functions and for the AUTO Trigger function.

This page is intentionally left blank.

Index

!

1 Mohm 10-5
50 ohms 10-5

A

Abort measurement 18-5, 19-6
AC coupling 10-4
AC|DC 2-2
Address switches 1-6
Analog Out
  Accuracy 5-6
  Description 5-5
  Enable 5-3
  Scaling factor 5-4
Aperture 13-5
Arbitrary block data 16-10
ARM-TRIG model 18-20
Arming 18-6
  Bus arm mode 18-7
  Delay by events 18-10
  Event count range 18-10
  Start delay 18-9
  Start slope 18-11
  Start source 18-12
  Stop delay 18-14
  Stop delay by events 18-15
  Stop delay range 18-14
  Stop slope 18-16
  Stop source 18-17
  Wait for bus 14-29
Array . . . . . 11-5, 11-7, 19-5 - 19-7, 19-10
  Example 20-12
  Fetch 11-11
  Read 11-9
ASCII
  Data format 9-3, 19-11
  Fixed format 9-4
Attenuation 2-2, 10-3
Auto
  Attenuation 10-3
  Gated voltage mode 13-17
  Levels selected by 10-7
  Once 10-7
  Power-on clearing 14-15
  Trigger on/off 10-7
Auto calibration on/off 7-3
Autotrigger 2-2, 10-7, 13-17
  Attenuation 10-3
  Off 10-6
Average
  Enable 6-3
  Event count 13-13
  Measurements 13-11
  Mode 13-13
  On/off 2-2, 13-11
  Samples in averaging 13-12
  Time mode 13-13

B

Big-indian 20-22
Block
  Arming 18-13
  Data 3-20, 9-3, 19-11
  Measurements 19-12
Boolean 3-19
Burst
  Carrier Frequency 11-16
Bus arming 18-8
  Exit 18-6
  Mode 18-7
  On/off 18-7
Bus drivers 1-8
Bus initialization 3-24

C

C-input
  Prescaling factors 13-19
CALCulate block 4-4

Calculations
Enable 6-3
Mathematics 6-6
Real time 19-11
Calculated data
Reading 6-4
Calibration 7-3, 14-29, 16-10, 16-14, 19-12
Channel
Selecting 13-4, 13-19
List 3-19
Character data 3-20
Check 2-2, 17-3
Clear Status 14-12
Clearing
status registers 14-15
CLS 3-25
CME 14-13, 14-23
Colon 3-12, 3-17
Command Error . . . . . 3-5, 14-13, 14-23, ... 20-14, 21-2, 21-18
Command header 3-17
Command List
Alphanumeric . . . . . 22-3 - 22-4, 22-14
Command tree . . . . . 3-13, 3-16 - 3-17
Commands
:ABORt 18-5
:ACQuisition:APERture 13-5
:Acquisition:HOFF 13-6
:Acquisition:HOFF:ECOunt 13-9
:Acquisition:HOFF:MODE 13-7
:Acquisition:HOFF:TIMe 13-8
:Acquisition:RESolution 13-10
:ARM 18-6
:ARM:SEQuence 18-6
:ARM:SEQuence1:LAYer1 :ECOunt 18-10
:ARM:SEQuence1:LAYer1:SLOPe 18-11
:ARM:SEQuence2:DELAy 18-14
:ARM:SEQuence2:ECOunt 18-15
:ARM:SEQuence2:SLOPe 18-16
:ARM:SEQuence2:SOURce 18-17
:ARM:SEQuence:LAYer1:COUNt . 18-8
:ARM:SEQuence:LAYer1:DELay . . 18-9
:ARM:SEQuence:LAYer1 :SOURce 18-12
:ARM:SEQuence:SOURce 18-7
:ARM:STARt 18-6
:ARM:STARt:LAYer1:COUNt 18-8
:ARM:STARt:LAYer1:DELay 18-9
:ARM:STARt:LAYer1:ECOunt . . . 18-10
:ARM:STARt:LAYer1:SLOPe 18-11
:ARM:STARt:LAYer1:SOURce . . 18-12
:ARM:STARt:SOURce? 18-7
:ARM:STOP:DELAy 18-14
:ARM:STOP:ECOunt 18-15
:ARM:STOP:SLOPe 18-16
:ARM:STOP:SOURce 18-17
:AVERage:COUNts 13-12
:AVERage:MODE 13-13
:AVERage:STATe 13-11
:CALCulate:AVERage:COUNt 6-10
:CALCulate:AVERage:STATe 6-9
:CALCulate:AVERage:TYPE 6-9
:CALCulate:DATA 6-4
:CALCulate:IMMediate 6-5
:CALCulate:LIMit:FAIL 6-16
:CALCulate:LIMit:LOWer:STATe . . 6-15
:CALCulate:LIMit:UPPer:STATe . . . 6-13
:CALCulate:LIMit[:STATe] 6-11
:CALCulate:MATH:STATe 6-7
:CALCulate:STATe 6-3
:CALibration:INTerpolator:AUTO . . . 7-3
:COMMunicate:GPIB:ADDRess . . . 16-3
:CONFigure 11-6
:CONFigure:ARRay 11-7
:CONFigure:DCYCle 11-22
:CONFigure:FREQuency 11-14
:CONFigure:FREQuency:BURSt :CARRier 11-16
:CONFigure:FREQuency:BURSt :PRF 11-17
:CONFigure:FREQuency:RATio . . 11-15
:CONFigure:FTIMe 11-23
:CONFigure:MAXimum 11-24
:CONFigure:MINimum 11-24
:CONFigure:NWIDth 11-21, 11-23
:CONFigure:PDUTycycle 11-22
:CONFigure:PERiod 11-18
:CONFigure:PHASe 11-20
:CONFigure:PTPeak 11-24
:CONFigure:PWIDth 11-21
:CONFigure:RTIMe 11-23
:CONFigure:TINTerval 11-19
:CONFigure:TOTalize :ACCumulated 11-26
:CONFigure:TOTalize :CONTinuous 11-29
:CONFigure:TOTalize:GATed . . . 11-25
:CONFigure:TOTalize:SSTop 11-27
:CONFigure:TOTalize:TIMed 11-28
:CONFigure:VOLTage:MAXimum 11-24
:CONFigure:VOLTage:MINimum . 11-24
:CONFigure:VOLTage:PTPeak . . 11-24
:DISPlay:ENABle 8-3
:FETCh:ARRay? . . . . . 11-10 - 11-11
:FETCh? 11-10

:FORMat:FIXed 9-4
:FREQuency:RANGe:LOWer . . . 13-14
:FUNCtion 13-4
:INITiate:CONTinuous 18-4
:INITiate:IMMediate 18-3
:INPut2:COMMon 10-10
:INPut:ATTenuation 10-3
:INPut:COUPling 10-4
:INPut:FILTer 10-9
:INPut:IMPedance 10-5
:INPut:LEVel 10-6
:INPut:LEVel:AUTO 10-7
:INPut:SLOPe 10-8
:MEASure:ARRay? 11-5
:MEASure:DCYCle? 11-22
:MEASure:FREQuency:BURSt
:CARRier? 11-16
:MEASure:FREQuency:BURSt
:PRF? 11-17
:MEASure:FREQuency:RATio? . 11-15
:MEASure:FREQuency? 11-14
:MEASure:FTIMe? 11-23
:MEASure:MAXimum? 11-24
:MEASure:MINimum? 11-24
:MEASure:NWIDth? 11-21
:MEASure:PDUTycycle? 11-22
:MEASure:PERiod? 11-18
:MEASure:PHASe? 11-20
:MEASure:PTPeak? 11-24
:MEASure:PWIDth? 11-21
:MEASure:RTIMe? 11-23
:MEASure:TINTerval? 11-19
:MEASure:TOTalize
:ACCumulated? 11-26
:MEASure:TOTalize
:CONTinuous? 11-29
:MEASure:TOTalize
:GATed? . . . . . 11-25 - 11-26
:MEASure:TOTalize:SSTop? . . . 11-27
:MEASure:TOTalize:TIMed? 11-28
:MEASure:VOLTage:MAXimum? 11-24
:MEASure:VOLTage:MINimum? . 11-24
:MEASure:VOLTage:PTPeak? . . 11-24
:MEASure? 11-4
:MEMory:DELete:MACRo 12-3
:MEMory:FREE:MACRo? 12-4
:MEMory:NSTates? 12-5
:READ:ARRay? 11-9
:READ? 11-8
:ROSCillator:SOURce 13-15
:SENSe:Acquisition:APERture . . . 13-5
:SENSe:Acquisition:HOFF 13-6
:SENSe:Acquisition:HOFF:ECOunt 13-9
:SENSe:Acquisition:HOFF:MODE . 13-7
:SENSe:Acquisition:HOFF:TIMe . . 13-8
:SENSe:Acquisition:RESolution . 13-10
:SENSe:AVERage:COUNts 13-12
:SENSe:AVERage:MODE 13-13
:SENSe:AVERage:STATe 13-11
:SENSe:FREQuency:RANGe
:LOWer 13-14
:SENSe:FUNCtion 13-4
:SENSe:ROSCillator:SOURce . . . 13-15
:SENSe:TOTalize:GATE:[STATe] 13-16
:SENSe:VOLTage:GATed:STATe 13-17
:STATus:DREGister0:ENABle 14-9
:STATus:DREGister0[:EVENt]? . . 14-10
:STATus:OPERation:CONDition? . 14-3
:STATus:OPERation:ENABle 14-4
:STATus:QUEStionable:[EVENt]? 14-8
:STATus:QUEStionable:CONDition? 14-6
:STATus:QUEStionable:ENABle . . 14-7
:SYSTem:ERRor? 16-4
:SYSTem:KEY 16-5
:SYSTem:PRESet 16-6
:SYSTem:SET 16-7
:SYSTem:TOUT[:STATe] 16-8
:SYSTem:UNPRotect . . . . 16-9 - 16-10
:TEST:SELect 17-4
:TOTalize:GATE 13-16
:TRIGger[:SEQuence1]:COUNt . . 18-13
:TRIGger[:STARt]:COUNt 18-13
:VOLTage:GATed:STATe 13-17
SOC? 14-3
SOEn 14-4
SOEv? 14-5
*CLS 14-12
*DMC 12-9
*EMC 12-10
*ESE 14-13
*ESR? 14-14
*GMC? 12-11
*IDN? 16-12
*LMC 12-12
*LRN? 12-8
*OPC 15-2
*OPC? 15-3
*OPT? 16-13
*PMC 12-13
*PSC 14-15
*PUD 16-14
*RCL 12-7
*RST 16-15
*SAV 12-6
*SRE 14-16

*STB? 14-17
*TRG 18-18
*TST? 17-5
*WAI 15-4
Common commands 3-12
Common via A. 2-2, 10-10
Configure 4-6, 11-6, 19-5
Array 11-7
Description 11-31
Continuously Initiated 18-4
Control function 1-7
Controller synchronization 19-2
Conventions . . . . . 1-4 - 1-5
Coupling 10-4
Cutoff frequency 10-9
CW 11-16

D

Data
Format 9-4, 19-11
Internal format 13-18
Type 9-3
DC coupling. 10-4
DCL. 3-7, 3-24
DDE. 14-13, 14-24
DDT. 3-25
Deadlock 3-6
Decimal data 3-18
DECL.BAS 20-4
Declaration
For GPIB interface 20-4, 20-30
Default settings 2-2, 16-15
Presetting the counter 16-6
Deferred commands 3-7
Define Macro. 12-9
Delay
After external start arming 18-9
After external stop arming 18-14
Delete one macro 12-3
Device
Clear 1-8, 3-7, 3-24
Initialization 3-24
Setup 12-8
Status. 14-16
Trigger 1-8
Device Dependent Error . . . 14-13, 14-23
Device Specific errors . . 3-5, 21-14, 21-18
Standardized 21-11
Device Status Register
Event 14-10
No. 0 14-31
Display
Enable 2-2, 8-3
State. 8-3
Double Precision floating point format . 9-3
Double quotes 3-20
DREG0 14-17
Dumping measurement results 19-12
Duration
See Pulse width
Duty Cycle measurements. 11-22
Duty factor 13-18

E

EAV . . . . . 14-16 - 14-17, 14-32, 16-4
Enable
Display 8-3
Macros 12-10
Mathematics. 6-7
Statistics. 6-8
Equality 1-4
Error
ASCII description 16-4
Available 14-16
Command. 21-2
Device specific 21-14
Execution 21-7
In self test. 17-5
Message available 14-32
Numbers, Standard 21-2
Query 21-12
Queue. 14-32, 21-17
Queue, Clearing 3-23
Reporting . . . . . 3-23, 21-17 - 21-19
Standardized device-specific 21-11
Standardized numbers 21-18
ESB . . . . . 14-13, 14-16 - 14-17, 14-19
Escape from erroneous conditions . . . 3-8
Event
Detection 18-21
Detection Layers Flow diagram . . 18-23
Event registers
Clearing 14-12
Event Status 14-16
Bit. 14-13
Register 14-14
Events
Before Start Arming 18-10
Before Stop Arming 18-15
Example language. 1-5
EXE 14-13, 14-24
Execution Control 3-5
Execution Error 3-5, 14-13, 14-24, 20-14,
. . . . . 21-7, 21-18

Expressions 1,4 6-6
Expression data 3-19
Ext. ref. 13-15
External reference 13-15
On/off 2-2
External Stop Arming
Delay 18-14

F

Fall time 11-23, 13-12, 13-18
Fast sampling . . . . . 20-12 - 20-13
Fetch 4-6, 19-6
Array 11-11
Description 11-32
One Result 11-10
Several measurement results . . . 11-11
FIFO 3-23
Filter 2-2, 10-9
Fixed data format 9-4
Format
Internal data 13-18
Formula
Mathematics 6-6
Macro 12-4
Frequency
Measurement 11-14
Profiling 20-10
Ratio measurements 11-15
Front panel memories 12-6
Function 2-3

G

Gate
On/off 13-16
Time 13-5
Gated Voltage Measurement 13-17
GET 18-7, 18-18, 18-22, 19-9
Get Macro 12-11
GPIB address 1-6, 16-3
Group Execute Trigger 18-18

H

Header path 3-17
Header separator 3-12
High Speed Voltage Measurements 13-14
Hold off 2-2
Event count 13-7, 13-9
Event range 13-9
Mode 13-7
On/off 13-6
Setting time 13-8
Time mode 13-7
Time range 13-8

I

Identification query 16-12
Idle and initiated layer flow diagram 18-21
Idle state 18-20
IFC 3-24
Immediate mode 18-7
Impedance 2-2, 10-5
Initialization . . . . . 3-24 - 3-26
Initiate 4-6, 19-5
Continuous 18-21, 19-6
Continuously 18-4, 19-9
Description 11-32
Immediate 18-20
Measurement 18-3
Initiated state 18-20
Input
Attenuation 10-3
Coupling 10-4
Impedance 2-2, 10-5
Selecting 13-19
Selecting channel 13-4
Summary 10-11
INPut block 4-3
Instrument model 4-2
Interface clear 3-24
Internal format 13-18
Internal reference 13-15
Interpolator calibration 7-3
Interrupted 3-6

K

K, L and M 6-6
Key 16-5
Keywords 3-18

L

Leading zeroes 9-4
Leaf node 3-17
Learn Device Setup 12-8
Learn Macro 12-12
Limit
Check lower 6-15
Check upper 6-13
Fail 6-16
Set lower 6-14
Set upper 6-12
Limit monitoring 14-31, 20-8
Enable 6-3, 6-11

Fail 6-16
Limit passed. 14-10
Set upper limit 6-12
Limit testing
Example. . . . . 20-8 - 20-9
Listener function 1-8
Local 1-6
Control 1-6
Lockout 3-8
Operation 3-8
Longform 1-5, 3-13
Low Pass Filter 10-9
Lower case 3-13
Lower Limit
Check. 6-15
Fail 6-16
Set 6-14

M

Macro . . . . . 12-16 - 12-18
Data types 12-15
Define 12-9
Delete 12-3
Delete all 12-13
Description. 12-15
Enable 12-10
Get. 12-11
How to execute 12-17
Learn 12-12
Memory states 12-5
Names 12-15
Purge. 12-13
Remove all. 12-13
Mathematics 2-3
Enable 6-3, 6-7
Select expression. 6-6
MAV . . . . . 3-7, 14-16 - 14-17, 14-19
Max. 3-18, 6-9, 20-6
Mean 6-9, 20-6
Measure 4-6, 19-7, 19-9
Array 11-5
Description. 11-31
Once 11-4
Measurement
Abort 18-5
Basic method. . . . . 19-8 - 19-10
Continuously initiated. 18-4
Function 4-5
Function Block 4-3
Initiate 18-3
No. of, on ext arm start 18-13
No. on each bus arm 18-8
Started (MST). 14-29
Stopped (MSP). 14-29
Status. . . . . 14-3 - 14-4
Terminate 19-6
Terminating 18-5
Trigger 18-18
Measuring
Burst CW 11-16
Duty cycle. 11-22
Fall time 11-23
Frequency 11-14
Frequency ratio 11-15
Function selection 11-6
Functions, Summary 11-3
Input selection 13-19
Period. 11-18
Phase. 11-20
PRF 11-17
Pulse width. 11-21
Rise time 11-23
Selecting function. 13-4
Time interval. 11-19
Totalize X gated by Y 11-25
Transition time 11-23
Voltage. 11-24
Measuring time 2-2, 13-5
Range. 13-5
Setting 13-5
Memory 2-3
Free for Macros 12-4
States. 12-5
Message
Available 14-16, 20-14
Exchange Control. 3-5
Exchange initialization 3-24
Terminator 3-6
Min. 3-18, 6-9, 20-6
Mnemonics 1-5, 3-12
Monitor
Limits 6-11, 20-8
Low limit 14-31
Delay trig 14-31
High limit. 14-31
MSP. 14-29
MSS. 14-17
MST. 14-29
Multiple measurements
See Array
Multiple queries 3-14

N

Negative slope 10-8
Non-decimal data 3-20
Notation habit 3-13
NRf 3-18
Numeric data 3-18
Numeric expression data 3-19

O

OFL 14-30
OPC 14-24
Operation Complete . . 14-13, 14-24, 15-2
Operation status 14-16, 14-29
  Condition 14-3
  Enable 14-4
  Event 14-5
  Group 14-27
OPR . . . . . 14-16 - 14-17
Option Identification 16-13
Optional nodes 3-16
Optionality 1-4
Output queue . . . . . 14-31 - 14-32
Overflow 14-30
  Status 14-6
  Message 3-23

P

Packed
  Data format 13-18
Parallel poll 1-8
Parameter list 3-19
Parenthesis 3-19
Parser 3-4
Peak to peak measurements 11-24
Period measurements 11-18
Phase measurements 11-20, 13-18
PMT 3-11, 3-17
PON 14-13, 14-23
Positive slope 10-8
Power ON 14-13, 14-23
  Status Clear 14-15
Prescaling factors 13-19
Preset 2-2, 16-6
  Status at power-on 14-15
  Status registers 14-11
PRF 11-17
Profiling 20-10
Program message terminator . . 3-11, 3-17
Program messages 3-10
Pulse
  Repetition Frequency 11-17
Pulse width 11-21, 13-12, 13-18
Purge Macro 12-13

Q

QUE . . . . . 14-16 - 14-17
Queries
  Multiple 3-14
Query Error . . . . . 3-5 - 3-6,
  . . . . 14-13, 14-24, 20-14, 21-12, 21-18
Questionable Data/signal 14-16
  Condition 14-6
  Enable 14-7
  Event 14-8
  Status 14-30
Quotes 3-20
QYE 14-13, 14-24

R

Ratio 11-15
Read 1-5, 4-6, 19-6, 19-8
  Array 11-9
  One Result 11-8
Read or Send Settings 16-7
Real
  Data format 13-18, 19-11
  Data format example 20-22
  Data type 9-3
Recalculate Data 6-5
Recall 2-3, 12-7
Reference
  Selection 13-15
Remote 1-6
Remote operation 3-8
Remote/local, ... 1-8
Remove Macro command 12-3
Remove macros 12-13
Repetition 1-4
Request Control 14-13, 14-24
Request Service 14-16
Reset 2-2, 3-24, 16-15
Resolution 13-10, 19-11
Response 3-15
  Data format 9-4
  Data type 9-3
  Message 3-6, 3-10
  Message terminator 3-14
Result
  Reading 11-8
Retrieve
  Front panel setting 12-7

Measurement data 19-6
Measurement result 11-10
Rise time 11-23, 13-12, 13-18
Rise/fall
Trigger levels 10-7
Rmt 3-14
Root level 3-13
Root node 3-13
RQC 14-13, 14-24
RQS 14-16, 14-19
RST 3-24

S

Sample Size
Statistics 6-10
Save 2-3, 12-6
Scaling factor
analog output 5-5
SCPI 3-2
SCPI compliance 16-11
SDC 3-7, 3-24
SDEV 6-9
Selective Device Clear 3-7, 3-24
Self Test 17-4, 17-5
Semicolon 3-12
SEND 1-5
SENSe block 4-3
Sensitivity
Trigger level 10-6
Sequential commands 3-7
Serial Poll 20-14
Service Request 1-8, 3-23, 20-14
Enable 14-16
Settings
Reading 16-7
Shortform 1-5, 3-13
Single 2-2
Measurements 13-11
On/off 13-11
Single quotes 3-20
Slope 2-2, 10-8
Arming start 18-11
Stop arming 18-16
Source
Start arming 18-12
Stop arming 18-17
Speed
Fast sampling 20-12
High speed Volt 13-14
Individually sync. meas. 19-9
Resolution 13-10
Summary . . . . . 19-13 - 19-14
Standard deviation 6-9, 20-6
Standard Event Status
Enable 14-13
Register 14-23
Standardized Device-specific errors 21-11
Standardized Error numbers 21-18
Start arming
Delay 18-9
Delay by events 18-10
Slope 18-11
Start source
Arming 18-12
Start/stop
Totalize 13-16
Statistics 2-3
Enable 6-3, 6-8
Example . . . . . 20-6 - 20-7
Reading results 6-4
Recalculating data 6-5
Sample size 6-10
Type 6-9
Status
Clear 14-12
Data structure 7 14-27
Enable reporting 14-11
Enabling Standard Event Status . . 14-13
Event Status Register 14-14
Limit monitor 14-10
Measurement started 14-4
Measurement stopped 14-4
Operation event 14-5
Overflow 14-6
Preset 14-11
Questionable Data/signal 14-6
Questionable Data/signal, Event . . 14-8
Register structure 14-18
Time out 14-6
Unexpected parameter 14-6
Using the reporting 14-18
Waiting for bus arming 14-4
Waiting for triggering 14-4
Status byte 3-21
Bit 0 14-10
Bit 2 14-32
Bit 3 . . . . . 14-6 – 14-8
Bit 5 . . . . . 14-13 - 14-14
Bit 6 14-17
Bit 7 . . . . . 14-3 - 14-5
Query 14-17
Reading 14-17, 14-21
Status data structures
Clear 3-25
Status reporting . . . . . 3-21 - 3-22, 14-18

Example . . . . . 20-14 - 20-17
Stop
Measurement 19-6
Stop Arming
Delay 18-14
Events 18-15
Slope 18-16
Source 18-17
Store
Front panel settings 12-6
String data 3-20
Subnodes 3-13
Suffixes 3-19
Summary
Instrument states 19-3
Measurement commands 11-33
Of command syntax 3-20
Of input amplifier settings 10-11
Speed . . . . . 19-13 - 19-14
Syntax 3-20
Swap A-B 2-2
Synchronization 19-2, 19-8
Syntax
Summary 3-20
Syntax and Style . . . . . 3-11, 3-13 - 3-15
System Version 16-11

T

Talker function 1-7
Terminate
Measurement 19-6
Terminating a measurement 18-5
Terminator 3-12
Test
Activating 17-5
Selecting internal self-test 17-4
signal 17-3
Time
Hold Off 13-8
Rise/fall 13-12, 13-18
Selecting measuring time 13-5
Time base
External/internal 13-15
Time interval 11-19, 13-12, 13-18
Time out for measurement (TIO) . . 14-30
Timeout
On/off 16-8
Range 16-9
Set 16-9
Status 14-6
TIO 14-30
Totalize
Gate on/off 13-16
X gated by Y 11-25
X gated by Y, accumulated 11-26
Start/stop 2-2
X start/stop by Y 11-27
X–Y Manually . . . . . 11-29 - 11-30
X–Y During a Preset Time 11-28
Transition filters 14-27
Transition time 11-23, 13-12, 13-18
Trigger levels
Fixed levels 2-2, 10-6
Levels selected by auto 10-7
Set levels 10-6
Slope 10-8
Trigger measurement . 18-7, 18-18, 18-22
Configurations 18-22
No. of, on ext arm start 18-13
Status 14-10
Subsystem 6-8
Subsystem states 19-4
System . . . . . 18-19, 18-21 - 18-24
Truncation rules 1-5

U

UEP 14-30
Unexpected parameter 14-30
Status 14-6
Unit separator 3-12
Unprotect 16-10
Unterminated 3-6
Upper case 3-13
Upper Limit
Check 6-13
Fail 6-16
Set 6-12
URQ 14-13, 14-23
User data 16-10
User Request 14-13, 14-23

V

Variable hysteresis
Auto levels 10-7
Voltage
Gated measurements 13-17
High speed 13-14
Voltage measurements 11-24

W

WAI 4-5
Wait for bus arming (WFA) 14-29
Wait-to-continue 15-4

Waiting for bus arming
Status. 14-4
Waiting for trigger and/or external arming
. 14-29
Waiting for triggering
Status. 14-4
WFA 14-29
WFT 14-29

X

X 3-12
X1/X10 attenuation 10-3
Xn-1 2-3
XOLD 6-6